STAT535: Statistical Computing **Fall 2020**

Problem Set 1: Functions And Flow Control

P. Flaherty

Objective This problem set will introduce you to control flow and functions in python. The problem set has three problems and you should save your code for each problem according to the scheme outlined in logistics. Don't forget to follow the style guidelines and include comments to help us understand your code.

Collaboration You may work with other students. However, each student must writeup and hand in their assignment individually. Be sure to indicate whom you worked with in the comments of your submission.

Readings Read the style guide.

Logistics The first problem should be named `ps1a.py`, the second problem should be named `ps1b.py`, and so on. You should upload all three to the gradescope site in the single upload box.

1. **House Hunting**

   You have graduated with a job and you want to start saving to buy a house. We are going to write a program to determine how long it will take to save enough money to make a down payment given the following assumptions:

   1. Call the cost of your house `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 with an annual return of `r` compounded monthly. At the end of each month, you receive an additional `current_savings*r/12` to put into your savings. Assume 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 the portion `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 to calculate how many months it will take you to save enough money for a down payment. You will want your main variables to be floats, so you should cast user inputs to floats. You should break your program down into two functions: `main()` and `compute_months(...)`. The `main` function is responsible for gathering the user input, executing the `compute_months`

function and print the result. The `compute_months(...)` function is should take the variables entered and return the number of months.

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

- The starting annual salary (`annual_salary`).
- The portion of salary to be saved (`portion_saved`).
- The cost of your house (`total_cost`).

Hints

- Look at `input()` if you need help with getting user input. For this problem set, you can assume the users will enter valid input.
- Initialize some state variables. You should decide what information your need. Be careful about values that represent annual amounts and those that represent monthly amounts.

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

Listing 1: Test Case 1

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

Listing 2: Test Case 2

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

2. **Saving with a raise**

We are going to build on your solution to part 1 by factoring in a raise every six months.

In `ps1b.py`, copy your solution to part 1 and modify 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 sixth month, increase your salary by that percentage. Do the same for the twelfth, eighteenth and so on.

Write a program to calculate how many months it will take you to save enough money for a down payment. Assume your investments earn a return of `r=0.04` and the required down payment fraction is 0.25. You should break your program down into two functions: `main()` and `compute_months(...)`. The `main` function is responsible for gathering the user input, executing

the `compute_months` function and print the result. The `compute_months(...)` function is should take the variables entered and return the number of months.

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

- The starting annual salary (`annual_salary`).
- The portion of salary to be saved (`portion_saved`).
- The cost of your house (`total_cost`).
- The semi-annual saraly raise (`semi_annual_raise`)

Hints Here is a rough outline of the stages your code should execute

- Retrieve the user input.
- Initialize some state variables. You should decide what information your need. Be careful about values that represent annual amounts and those that represent monthly amounts.
- Be careful about when you increase the salary - this should happen **after** month 6, 12, 18 and so on.

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

Listing 3: 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 house: 500000
Enter the semi-annual raise, as a decimal: .03
Number of months: 142
```

Listing 4: 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 house: 800000
Enter the semi-annual raise, as a decimal: .03
Number of months: 159
```

Listing 5: 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 house: 1500000
Enter the semi-annual raise, as a decimal: .05
Number of months: 261
```

3. **Finding the right amount to save**

   Now, we're going to answer the question of what we can afford after saving for three years. Here are your assumptions

   1. Your semi-annual raise is 0.07
   2. Your investments have an annual return of 0.04
   3. The down payment portion is 0.25
   4. The cost of the house is $1M

   You will find the lowest rate of savings to achieve the down payment in 36 months. You only need to be within $100 of the required down payment.

   Write a program to calculate the lowest savings rate, as a function of starting salary. You should use **bisection search** to help you do this efficiently. Keep track of the number of steps it takes for your search to finish.

   Because we are searching for a floating point value, we are going to limit ourselves to two decimal points of accuracy. We can search for an integer between 0 and 10000 (using integer division), and then convert it to a decimal percentage (using float division) to use when we are calculating `current_savings` after 36 months. By using this range, there are only a finite number of numbers that we have to search over – and this will help prevent infinite loops. You code should print out a decimal (e.g. 0.0704 for 7.04%).

   Try different inputs for your starting salary and see how the percentage you need to save changes. If it is not possible to save the entire down payment in 36 months, your function should notify the user that it's not possible. **Make sure your program prints the results in the format shown in the test cases below.** The function `compute_rate` should return `None` for the rate and number of bisection steps.

   You should break your program down into two functions: `main()` and `compute_rate(...)`. The `main` function is responsible for gathering the user input, executing the `compute_rate` function and print the result. The `compute_rate(...)` function is should take the variables entered and return the rate and number of bisection steps.

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

   - The starting annual salary (`annual_salary`).

   Hints

   - There are multiple right ways to implement bisection search so your results may not match the test cases exactly.
   - Depending on your stopping condition and how you compute a trial value for search, your number of steps may vary slightly from the example.
   - Watch out for integer division when calculating if a percentage saved is appropriate and when calculating final decimal savings rates.
   - Remember to reset the appropriate variables to the initial values for each iteration of bisection search.

Listing 6: Test Case 1

```
>>>
Enter your starting annual salary: 150000
Minimal savings rate: 0.4411
Steps in bisection search: 12
>>>
```

Listing 7: Test Case 2

```
>>>
Enter your starting annual salary: 300000
Minimal savings rate: 0.2206
Steps in bisection search: 9
>>>
```

Listing 8: Test Case 3

```
>>>
Enter your starting annual salary: 10000
It is not possible to pay the down payment in three years.
>>>
```