Problem Set 1: Compound Interest

Pset Buddy

You do not have a buddy assigned for this pset.

PLEASE DON'T DELETE ANY OF THE COMMENTS IN THE TEMPLATE PROVIDED AND USE THE VARIABLE NAMES SPECIFIED BELOW TO GET FULL CREDIT!

1) Introduction

1.1) Objectives

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

1.2) Overview

- Complete each of the **three** problems in the respective python files: **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.
- **ps1_tester.py** is a tester file that you may run to test your code. **Make sure ps1_tester.py is in the same folder as ps1a.py, ps1b.py, ps1c.py, and put_in_function.py.**

1.3) Collaboration

- Students may work together, but each student should write up and hand in their assignment separately. Students may not submit the exact same code.
- Buddies may work together and submit the same code.
- Students are not permitted to look at or copy each other's code or code structure.
- Include the names of your collaborators in comment at the start of each file.
- **Please refer to the collaboration policy in the Course Information for more details.**

1.4) Important Notes

- Read the Style Guide sections 1, 2, 3, and 4.
- If you get an error like `ModuleNotFoundError: No module named 'ps1b_in_function'` at any point, restart the Spyder kernel.
- **Get user input and declare variables under the correct comment headers in ps1a.py, ps1b.py, and ps1c.py. Failure to do so (e.g. using different variable names where specified or declaring them under the wrong comment headers) will result in a 0 by the grader.**
- **Do not remove or change any of the comments that are provided in the original ps1a.py, ps1b.py, and ps1c.py files.**
- The comments at the top of the files in the pset download may refer to 6.100A. This is because we use the exact same files for both 6.100A and 6.100L.

2) Part A: Saving for a House

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 it takes to save up for a down payment**. The cost of your down payment is calculated by **multiplying the total cost of your dream house by the down payment percentage.**

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 yearly salary (**`yearly_salary`**)
2. The portion of salary to be saved (**`portion_saved`**). This variable should be in decimal form (e.g. 0.1 for 10%).
3. The cost of your dream home (**`cost_of_dream_home`**)

Writing the Program. You will need to determine how many months it will take given the following information:

1. `yearly_salary`, as described above.
2. `portion_saved`, as described above.
3. `cost_of_dream_home`, as described above.
4. `portion_down_payment`, the percentage of the total cost needed for a down payment. **Assume `portion_down_payment = 0.25 (25%)`.**
5. The amount that you have saved thus far is **`amount_saved`**, which starts at $0.
6. You get an annual rate of return **`r`**. In other words, at the **end of each month**, you receive an additional **`amount_saved * (r/12)`** funds for your savings (the 12 is because `r` is an annual rate). **Assume `r = 0.05 (5%)`.**
7. At the end of each month, your savings increase by (1) a percentage of your monthly salary and (2) the monthly return on your investment. **(Note: The investment amount used to calculate the monthly return is the amount you had saved at the start of each month.)**

Output.

1. Your program should store the number of months required to save for the down payment using a variable called **`months`**.

Notes

- Be careful about values that represent **annual** amounts versus **monthly** amounts.
- If the number of months your program returns is off by one, reread the **highlighted** text above.
- Assume that users enter valid inputs (e.g. no string inputs when expecting a float).
- Your program should print outputs in the same format as the test cases below. If you're using Spyder, you might see extra lines in between outputs. Don't worry about them!

2.1) Testing

2.1.1) Manual Test Cases

Test Case 1

```
Enter your yearly salary: 112000
Enter the percent of your salary to save, as a decimal: .17
Enter the cost of your dream home: 750000
Number of months: 97
```

Test Case 2

```
Enter your yearly salary: 65000
Enter the percent of your salary to save, as a decimal: .20
Enter the cost of your dream home: 400000
Number of months: 79
```

Test Case 3

```
Enter your yearly salary: 350000
Enter the percent of your salary to save, as a decimal: .3
Enter the cost of your dream home: 10000000
Number of months: 189
```

2.1.2) Student Tester

Run **ps1_tester.py** in the same folder as **ps1a.py** and **put_in_function.py**. You should pass the first 3 test cases.

3) Part B: Saving with a raise

In Part A, we assumed that your salary did not change over time. However, 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 **adding a salary raise every six months.** Copy over your solution from Part A into the corresponding sections in **ps1b.py.**

User Inputs. There is one additional user input in Part B. Remember to cast these inputs as floats and in the following order before declaring other variables.

1. The starting yearly salary (`yearly_salary`)
2. The portion of salary to be saved (`portion_saved`)
3. The cost of your dream home (`cost_of_dream_home`)
4. The semi-annual salary raise (`semi_annual_raise`), which is a decimal percentage (e.g. 0.1 for 10%)

Writing the Program. Write a program to calculate how many months it will take for you to save up for a down payment. You can reuse much of the code from Part A. Like before, assume that your investments earn an annual rate of return `r = 0.05` (or 5%) and that `portion_down_payment = 0.25` (or 25%). In this version, `yearly_salary` increases by `semi_annual_raise` at the **end of every six months.**

Output.

1. Like Part A, your program should store the number of months required to save up for your down payment using a variable called `months`.

Notes

- Like Part A, **the investment amount used to calculate the monthly return is the amount you had saved at the *start of each month*.**
- Be careful about values that represent **annual** amounts versus **monthly** amounts.
- Raises should only happen at the end of the 6th, 12th, 18th month, and so on.
- If the number of months your program returns is off by one, reread the **highlighted** text above.
- Assume that users enter valid inputs (e.g. no string inputs when expecting a float).
- Your program should print outputs in the same format as the test cases below. If you're using Spyder, you might see extra lines in between outputs. Don't worry about them!

3.1) Testing

3.1.1) Manual Test Cases

Test Case 1

```
Enter your starting yearly 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: 92
```

Test Case 2

```
Enter your starting yearly 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: 131
```

3.1.2) Student Tester

Run **ps1_tester.py** in the same folder as **ps1b.py** and **put_in_function.py**. You should now pass the first 5 test cases.

4) Part C: Choosing an Interest Rate

In Part A and B, you explored how (1) the percentage of your salary saved each month and (2) a semi-annual raise affects how long it takes to save for a down payment given **a fixed rate of return, `r`.**

In Part C, we will have a **fixed initial amount** and the ability to choose a value for the rate of return, `r`. Given an initial deposit amount, our goal is to find the **lowest** rate of return that enables us to save enough money for the down payment in 3 years.

precisamos encontrar a menor taxa de juros que faça nosso objetivo ser
atingido em no máximo tres anos.

User Inputs. Cast the user input 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 the minimum rate of return `r` needed in order to reach your goal of a sufficient down payment in 3 years, given an `initial_deposit`. To simplify things, assume:

1. The cost of the house that you are saving for is $800,000.
2. The down payment is 25% of the cost of the house.

Use the following formula for compound interest in order to calculate the predicted savings amount given a rate of return `r`, an `initial_deposit`, and `months`:

$$\texttt{amount\_saved} = \texttt{initial\_deposit} \times (1 + \frac{r}{12})^{\texttt{months}}$$

You will use bisection search to determine the **lowest** rate of return `r` that is needed to achieve a down payment on a $800,000 house in 36 months. Since hitting this exact amount is a bit of a challenge, we only require that your savings be within $100 of the required down payment. For example, if the down payment is $1000, the total amount saved should be between $900 and $1100 (exclusive).

a diferença pra mais e pra menos pode ser de até 100$

Your bisection search should update the value of `r` until it represents the lowest rate of return that allows you to save enough for the down payment in 3 years. `r` should be a **float** (e.g. 0.0704 for 7.04%). Assume that `r` lies somewhere between 0% and 100% (inclusive).

Outputs.

1. The variable **`steps`** should reflect the number of steps your bisection search took to get the best `r` value (i.e. `steps` should equal the number of times that you bisect the testing interval).
2. The variable **`r`** should be the lowest rate of return that allows you to save enough for the down payment in 3 years.

Notes

- There may be multiple rates of return that yield a savings amount that is within $100 of the required down payment on a $800,000 house. The grader will accept any of these values of `r`.
- **If the initial deposit amount is greater than or equal to the required down payment minus $100, then the best savings rate is 0.0.**
- **If it is not possible to save within $100 of the required down payment in 3 years given the initial deposit and a rate of return between 0% and 100%, `r` should be assigned the value `None`.**
  - **Note**: the value `None` is different than `"None"`. The former is Python's version of a null value, and the latter is a string.
- Depending on your stopping condition and how you compute the amount saved for your bisection search, your number of steps may vary slightly from the example test cases. Running **ps1_tester.py** should give you a good indication of whether or not your number of steps is close enough to the expected solution.
- If a test is taking a long time, you might have an infinite loop! Check your stopping condition.
- Your program should print in the same format as the test cases below. If you're using Spyder, you might see extra lines in between outputs. Don't worry about them!

4.1) Testing

4.1.1) Manual Test Cases

Test Case 1

```
Enter the initial deposit: 65000
Best savings rate: 0.380615234375 [or very close to this number]
Steps in bisection search: 12 [or very close to this number]
```

Test Case 2

```
Enter the initial deposit: 150000
Best savings rate: 0.09619140625 [or very close to this number]
```

```
Steps in bisection search: 11 [May vary based on how you implemented your bisection search]
```

Test Case 3

```
Enter the initial deposit: 1000
Best savings rate: None
Steps in bisection search: 0 [May vary based on how you implemented your bisection search]
```

4.1.2) Student Tester

Run **ps1_tester.py** in the same folder as **ps1c.py** and **put_in_function.py** and you should pass all 8 test cases.

5) Hand-in Procedure

5.1) Time and Collaboration Info

At the start of each file, in a comment, write down the names of your collaborators. For example:

```
# Problem Set 1A
# Name: Jane Lee
# Collaborators: John Doe
```

Please estimate the number of hours you spent on the Problem Set in the question box below.

5.2) Half-way Submission

All students should submit their progress by the half-way due date (1 week before the final due date).
This submission will be worth 1 point out of the problem set grade and will not be graded for correctness. The intention is to make sure that you are making steady progress on the problem set as opposed to working on it in the final days before the due date.

You may upload new versions of the pset files (choose any one) until Sep 21 at 09:00PM. You cannot use extensions or late days on this submission.

Select File No file selected

Submit

You have infinitely many submissions remaining.

5.3) Final Submit

Before you submit your files, remove all extra debugging print statements. In addition, open a new console in Spyder and rerun your programs.

Be sure to run the student tester and make sure all the tests pass. However, the student tester contains only a subset of the tests that will be run to determine the problem set grade. Passing all of the provided test cases does not guarantee full credit on the pset.

You may upload new versions of each file until Oct 05 at 09:00PM, 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. **Please note that we will be grading the your last submission.**

To submit a pset with multiple files, you may submit each file individually through the submission page. Be sure to press **Submit** when you are ready to submit your code.

And that's it! Congrats on finishing your first 6.100L pset :)

5.3.1) Part A

Select File No file selected

Submit

You have infinitely many submissions remaining.

5.3.2) Part B

Select File No file selected

Submit

You have infinitely many submissions remaining.

5.3.3) Part C

Select File No file selected

Submit

You have infinitely many submissions remaining.

6.100L Introduction to CS and Programming Using Python

Fall 2022