MTH 3300
Spring 2024

Programming Problem 2a: invest.py

Directions: Download the template files I have provided on Blackboard. Then open Spyder, load these template files, and write the following programs. Submit your source code via Gradescope in `.py` format. READ THE INSTRUCTIONS on submitting your work in the Course Documents section of Blackboard.

Specify collaborators/resources or explicitly specify that none were used in the comments at the top of your `.py` file. Failure to include this will result in a zero on the assignment.

Be sure to read the SPECIFICATIONS carefully! And write comments!

You are modeling the growth of an investment over time. You use different formulas depending on whether you have a *continuous* growth rate or a nominal growth rate compounded a certain number of times per year.

For continuous growth,

$$A = Pe^{kt}$$

where P is the initial value of the investment at time t = 0, k is the relative growth rate in percent per year (expressed as a decimal), t is the time elapsed in years, and A is the value of the investment at time t. Also, e is the base of the natural logarithm ($e \approx 2.718$).

Alternatively, given a nominal interest rate r (expressed as a decimal) and the number of times the interest is compounded per year, n,

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

is the formula that gives the value of the investment A after time t in years.

Write a program that prompts the user to select continuous or non-continuous compounding exponential growth (C for continuous and N for non-continuous exponential growth). (Assume that the user complies and does not enter any other option.) The program should then do the following **two** times: it will ask for a period in years, a growth rate, and the number of times compounded per year (if applicable); the program will then compute the value of the investment after that many years have elapsed, using the appropriate exponential growth formula provided above. This should be done *cumulatively* – that is, the end investment for the first iteration should be used as the initial investment for the second iteration.

So, for example, suppose that the user selects continuous growth and enters an initial amount of 300, a first period of 4 years, and a first growth rate of 1.2%. Then, the investment at the end of the first period would be 314.75119659734116 since

$$300 \cdot e^{(0.012)(4)} = 314.75119659734116.$$

Suppose then that the second period was 2.5 years, and the second growth rate is 5%; then the investment at the end of the second period will be

$$314.75119659734116 \cdot e^{(0.05)(2.5)} = 356.65983152520965.$$

That last number rounded to two decimal places, 356.66, is what the program should display as the final value.

So, when you run your program, it should look like this:

```
Enter C for continuous or N for noncontinuous: C
Enter the initial amount: 300
Enter the first time period in years: 4
Enter the first growth rate as a percentage: 1.2
Enter the second time period in years: 2.5
Enter the second growth rate as a percentage: 5
The ending value is 356.66
```

(The values after each `:`, like `C`, `300` and `4` and `1.2`, are user entries; the program should produce the rest.)

Alternatively, suppose the user selects noncontinuous and enters an initial amount of 300, a first period of 4 years, and a first growth rate of 1.2%, compounded 4 times a year. Then, the investment at the end of the first period would be 314.728580546 since

$$300\left(1+\frac{0.012}{4}\right)^{4*4} = 314.728580546$$

Suppose then that the second period was 2.5 years, and the second growth rate is 5%, compounded 12 times a year; then the investment at the end of the second period will be

$$314.728580546\left(1+\frac{0.05}{12}\right)^{12*2.5} = 356.541599895.$$

As previously mentioned, that last number rounded to two decimal places, 356.54, is what the program should display as the final value.

So, when you run your program, it should look like this (where again, the values after each `:` are user entries; the program should produce the rest.):

```
Enter C for continuous or N for noncontinuous: N
Enter the initial amount: 300
Enter the first time period in years: 4
Enter the first growth rate as a percentage: 1.2
Enter the number of times compounded per year: 4
Enter the second time period in years: 2.5
Enter the second growth rate as a percentage: 5
Enter the number of times compounded per year: 12
The ending value is 356.54
```

Hints: make sure you try calculating my sample values by hand before programming to make sure you understand the task! Finally, if you are tempted to figure out a way to get Python to "repeat itself two times" – don't do that; write similar code two times (we'll learn about repetition soon enough).

Specifications: your program must

- ask the user to pick continuous or noncontinuous growth
- ask the user to enter the initial amount.
- ask the user to enter years and a growth rate two times. In the case of non-continuous, it should also ask the user the number of times compounded per year. The program should accept the growth rates input as **percents** (but input without the percent sign – so "3.5%" will be input to the program as just `3.5`).
- compute the **final** amount as described above and displays it to two decimal places.

The output should be *exactly* as specified in the sample above. When I say exact, I mean identical, down to the spacing. In this class, I have written autograders for most assignments, which run on your code as soon as you submit it and let you know if you pass the tests I've written. Those tests check if your output matches what I've specified down to the spaces, so you must follow my specifications strictly. (This may seem annoying, but it is a valuable skill to master. It is widespread in computer science to have precise instructions for something to work or for a client to be happy.) The benefit of autograders is that you can immediately know if your code isn't doing exactly the right thing and fix it before the deadline!

On Gradescope, you can see the expected output and your code's output for the visible test cases. I suggest using text-compare.com to check for minor discrepancies.

Finally, any program with more than a few lines of code (nearly all of them) should have comments describing what blocks of the code do. We'll learn more about how to better organize code in the future, but for now, you should put in a couple of comments describing what blocks of code are doing so that someone reading your program can easily see what is happening. You should also ensure any variables you use are named well, and your header describes what the program does. Finally, be mindful of the spacing in your program: how can you use an extra line here or there to make it easier to read, but not too many? There are several correct answers to these style questions, so go with what you think makes it easiest to read and stay consistent. I will give feedback on your style to help you improve it.

Again, your input prompts must match the ones given above *exactly* (including spaces) to pass the test cases on Gradescope.

Note: This problem has two visible test cases on Gradescope. You will see if your code passes these two test cases. In addition, it has two invisible test cases, which you will not be able to see until after grades are released (you will not see if your code passed or did not pass these). So ensure you run additional tests to cover all the possible inputs described! (You must work out the correct output in your test cases.)