MTH 3300
Fall 2024

Programming Problem Set 1

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. You need not list me or our class notes as collaborators/resources. **Failure to include this will result in a zero on the assignment.**

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

Note: There are two different programming problems in this assignment. They each have their own submission area on Gradescope.

Assignment goals: Get practice with various numerical operations in Python, getting user input, and appropriately formatting output. Also, using the `math` and `random` modules and doing conversions from binary to decimal.

1) population.py

Suppose that you are a demographer who wants to model the growth of a nation's population over time. A simple model for this growth is the standard exponential model.

$$P = P_0 e^{rt}$$

where P_0 is the initial population at time t = 0, r is the relative growth rate in percent per year (expressed as a decimal), t is the time elapsed in years, and P is the population at time t. Also, e is the base of the natural logarithm ($e \approx 2.718$).

Write a program that prompts the user to enter a value for the initial population. The program should then do the following **two** times: it will ask for a period in years, and a relative growth rate; the program will then compute the new population after that many years have elapsed, using the population growth formula provided above. This should be done *cumulatively* – that is, the end population for the first iteration should be used as the initial population for the second iteration.

So, for example: suppose that the user enters an initial population of 300, a first period of 4 years, and a first growth rate of 1.2%. Then the population 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 population at the end of the second period will be

$$314.751 \cdot e^{(0.05)(2.5)} = 356.65983152520965.$$

That last number rounded to six decimal places, 356.659832, is what the program should display as the final population. (Don't worry that these numbers aren't integers.)

Finally, calculate the overall percentage growth from the initial population to the final population as follows:

$$\text{percentage growth} = \frac{\text{final population} - \text{initial population}}{\text{initial population}} \times 100$$

and display it at the end, rounded to three decimal places, along with the overall time period during which that percentage growth occurred (see sample run of the program below).

For the values above, the percentage growth would be

$$\frac{356.65983152520965 - 300}{300} \times 100 = 18.88661050840322$$

and your program should display 18.887.

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

```
Enter the initial population: 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 population is 356.659832
The overall growth in the 6.5 year period is 18.887 percent
```

(The numbers after each `:`, like `300` and `4` and `1.2`, are user entries; the rest should be produced by the program.)

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 three times" – don't do that; write similar code three times over (we'll learn about repetition soon enough).

Specifications: your program must

- ask the user to enter the initial population.
- ask the user to enter years and a growth rate two times. 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** population as described above and displays it to six decimal places.
- compute the percentage growth as described above and displays it to three decimal places, along with the total timer period over which it occurred.

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 make sure you run additional tests on your own to cover all the possible inputs described! (You must work out the correct output in your test cases.)

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

Hint: make sure you try calculating my sample values by hand before programming to ensure you understand the task!

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 exactly. (This may seem annoying, but it is a valuable skill to master. It is very common 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!

Finally, any program with more than a few lines of code (so nearly all of them) should have comments describing what blocks of the code do. We'll learn more about how to organize code better 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 going on. 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.

You can submit your code as many times as you wish before the deadline to fix any issues. If you have questions about what the autograder tells you is wrong, please ask!

2) binary.py

Recall that we briefly discussed the *binary system* (or *base two*) in class. A number is represented in binary by a sequence of `1`'s and `0`'s (also known as *bits*), which have place values given by powers of 2 instead of powers of 10. In a four-digit binary number, the left digit would be the 8's digit; the next digit would be the 4's digit, the third would be the 2's digit, followed by the 1's digit.

For example, `1101` in binary would represent the (base-10) number 13, because this number has, reading from the left, one 8, one 4, no 2, and one 1, and 8 + 4 + 1 = 13. And `0111` would represent the (base-10) number 7, since this number has no 8, one 4, one 2, and one 1.

Write a program that takes input from the user the 8's digit, 4's digit, 2's digit, and 1's digit of a 4-digit binary number and prints its base-10 equivalent. The program should then display those four digits in a row, together with the equivalent decimal number.

Next, your program should generate a random integer between 0 and 15 (inclusive) and print the boolean value (`True` or `False`) that results from checking if the binary number is less than the randomly generated number. Specifically, your program should use `randrange` from the `random` module to generate a single random number (let's call it `x`) between 0 and 15 inclusive. Then print the result of the comparison `num<x`, where `num` is the base-10 version of the binary number.

When you run your program, a sample run should look like this:

```
Enter 8's digit: 1
Enter 4's digit: 1
Enter 2's digit: 0
```

```
Enter 1's digit: 0
The binary number 1100 is 12 in base ten
Less than the randomly generated number 5? False
```

where `5` is randomly generated, or this:

```
Enter 8's digit: 0
Enter 4's digit: 0
Enter 2's digit: 1
Enter 1's digit: 1
The binary number 0011 is 3 in base ten
Less than the randomly generated number 9? True
```

where `9` is randomly generated.

Note that every time I run the program, I get a different number (between 0 and 15). (I have some code at the start of the template that you should comment out to test this, but make sure to leave it in when uploading to Gradescope! If you run the code in Spyder without commenting these lines out, you should see nothing.)

You should complete this program only using things we have discussed already in the class: **don't** use lists or tuples (these are sequences of variables enclosed in either `[]` or `()`), **don't** use loops, and definitely **don't** use any of Python's functions for binary (like `bin`).

Specifications: your program must

- Get the four digits of the binary number as input from the user. You may assume the user complies and enters only a `0` or a `1` for each of the binary digits. (You do *not* have to worry about the user entering something like `3` or `2.2` or `1.0` for the binary digits.)
- use `random.randrange` to generate a number between 0 and 15 inclusive. Display this randomly generated number as specified above in the sample runs.
- display the four digits, as well as the equivalent decimal number, in the manner shown above. In particular, I want to see binary number listed like `1100`, *not* like `(1, 1, 0, 0)` or `1 1 0 0`.
- display the comparison result between the binary and randomly generated numbers as specified and in the format above.
- only uses techniques that we have discussed in class thus far – in particular, **no use of the `bin` function, no use of loops, and no use of lists or tuples**.

Note: This problem has five visible test cases on Gradescope. You will see if your code passes these five test cases. In addition, it has three invisible test cases, which you will not be able to see at all until after grades are released (you will not see if your code passed or did not pass these). So make sure you run additional tests on your own to cover all the possible inputs described!

Your input prompts and output must match the format given above *exactly* (including spaces) to pass the test cases on Gradescope.

Failure to follow specifications will result in deduction of points, even if test cases pass.