MTH 3300
Spring 2018
E. Fink
February 5, 2018

Homework 2

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 to me via Blackboard, in `.py` format; do NOT send any other files. READ THE INSTRUCTIONS on how to submit your work in the Course Documents section of Blackboard.

Be sure to read the SPECIFICATIONS carefully for all problems! And write comments!

---

1) population.py

Suppose that you are a demographist, who wants to model the growth of the population of a nation 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. e, of course, 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 **three** times: it will ask for a number of 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, and the end population for the second iteration should be used as the initial population for the third iteration.

So, for example: suppose that the user enters an initial population of 300, a first time period of 4 years, and a first growth rate of 1.2%. Then the population at the end of the first time period would be 314.751, since

$$300 \cdot e^{(0.012)(4)} = 314.751.$$

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

$$314.751 \cdot e^{(0.05)(2.5)} = 356.66.$$

Finally, suppose that the user enters 1 year for the last time period, and 2.1% for the last growth rate; then the population at the end of the last time period will be

$$356.66 \cdot e^{(0.021)(1)} = 364.229.$$

That last number, 364.229, is what the program should display as the final population. (Don't worry about that fact that many of these numbers aren't integers.)

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

```
Enter initial population: 300
Enter first time period (in years): 4
Enter first growth rate (in percent): 1.2
Enter second time period (in years): 2.5
Enter second growth rate (in percent): 5
Enter third time period (in years): 1
Enter third growth rate (in percent): 2.1
The final population is 364.2288848868699
```

(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 with my sample values by hand before you start programming, to make sure you understand the task! Finally, in case you are tempted to figure out a way to get Python to "repeat itself three times" – don't do that, just write similar code three times over (we'll learn about repetition soon enough).

Specifications: your program must

- clearly ask the user to enter the initial population.
- ask the user three times to enter a number of years, and a growth rate. 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 clearly display it. You are not required to round this value to an integer.

***Challenge**: try to write this program using only three variables.*

---

2) triangle.py

Write a program that will ask the user to input *three positive numbers in descending order*. The program will then – using a specific procedure! – compute the three angles of the triangle that has those three numbers as sidelengths, in degrees.

You will do this by using the Law of Cosines to find the largest angle, then using the Law of Sines to find the middle angle, and then subtract to find the smallest angle. (There are other ways to proceed – you might say they are better ways! However, I am **insisting** that you do it in this manner.)

I strongly suggest you look up the Law of Sines and the Law of Cosines in case you've forgotten them. Also, don't forget about converting from radians to degrees. In addition, you should Google the trigonometric and inverse trigonometric functions that you will need for your calculations.

When you run your program, a sample run might look like this (where the user inputs `5`, `4` and `3`):

```
Enter largest side length: 5
Enter middle side length: 4
Enter smallest side length: 3
The angles are:
90.0
53.13010235415598
36.86989764584401
```

or this (where the user inputs `4.1`, `2.6` and `2.4`):

```
Enter largest side length: 4.1
Enter middle side length: 2.6
Enter smallest side length: 2.4
The angles are:
110.105509787916
36.548442826074876
33.34604738600913
```

Hints: the largest angle is opposite the largest side, the middle angle is opposite the middle side, and the smallest angle is opposite the smallest side. Also, be *very* careful about order of operations!

Specifications: your program must

- ask the user to enter three side lengths in descending order (two consecutive equal sides is valid also), each of which can be any positive number – even one with decimals. You may assume that the user obeys – if the user enters sides out of order, or negative numbers, then your program does not need to work. You may also assume that the user enters numbers that are the sidelengths of a real triangle! That is, you don't have to worry about the user entering `100`, `2` and `1`, because no real triangle has those three sidelengths (remember the triangle inequality?).
- correctly calculate the three angles in the triangle, in **degrees**, under the assumptions mentioned above.
- use the strategy I outlined above: Law of Cosines to find the largest angle, Law of Sines to find the middle angle, and subtracting to find the smallest angle.
- print out the three angles in degrees, **each on a separate line**. You don't have to worry about how many decimal places they are output with.

***Challenge**: make the program print a message if the side lengths entered are either not all positive, aren't entered in descending order, or don't satisfy the triangle inequality. You might need to read ahead to do this.*

---

3) 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 an eight-digit binary number, the left digit would be the 128's digit, the next digit would be the 64's digit, the third would be the 32's digit, followed by the 16's, 8's, 4's, 2's, and 1's digits.

For example, `01011101` in binary would represent the (base-10) number 93, because this number has, reading from the left, no 128, one 64, no 32, one 16, one 8, one 4, no 2, and one 1, and 64 + 16 + 8 + 4 + 1 = 93. And `00000111` would represent the (base-10) number 7, since this number has no 128, no 64, no 32, no 16, no 8, one 4, one 2, and one 1.

Write a program that takes **no input from the user**, but prints a random 8-digit binary number, and its base-10 equivalent. Specifically, your program should use the `random` module to pick 8 random numbers that should each be either `0` or `1`. The program should then display those 8 digits in a row, together with the equivalent decimal number.

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

```
Here's a random example of binary!
The binary number 1 0 0 0 1 1 1 0 is the same as the decimal number 142.
```

or this:

```
Here's a random example of binary!
The binary number 0 0 1 1 0 0 1 1 is the same as the decimal number 51.
```

Every time I run the program, I should get a different number (between 0 and 255).

You should be able to complete this program (and the challenge) 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`). If/else statements are acceptable, but unnecessary. Instead, use mathematics – multiplication, exponentiation, and maybe `//` and `%` (the last two might helpful for the challenge).

Specifications: your program must

- accept NO user input.
- use `random` to generate eight different random numbers – each should be 0 or 1.
- display the eight generated digits, as well as the equivalent decimal number, in the manner shown above. In particular, I want to see binary numbers listed like `1 0 1 0 1 1 0 0`, *not* like `(1, 0, 1, 0, 1, 1, 0, 0)`. (`10101100`, without the spaces in between, *is* acceptable.)
- display different numbers each time the program is run.
- only use 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**.

***Challenge**: after you answer this question, have the program continue by asking the user to type in a number between 0 and 255 in decimal, and then printing out the binary equivalent. For example, a run might look like this (where the user enters the value 24):*

```
Now, enter a decimal number between 0 and 255: 24
This number is equivalent to the binary number 0 0 0 1 1 0 0 0
```