MTH 3300
Summer 2018
E. Fink
June 11, 2018

Homework 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 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) interest.py

When depositing a principal of P dollars into a savings account with an annual interest rate of r percent (where r is expressed as a decimal), the account balance A when interest is compounded continuously is given by the formula

$$A = Pe^{rt},$$

where t is in years, and 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 principal. You will then ask the user for 5 interest rates: the rate for years 0 to 2, the rate for years 2 to 4, for years 4 to 6, for years 6 to 8, and for years 8 to 10. Your program will then calculate the account balance of an account started with the given principal, subject to these continuously compounded interest rates for the given periods of time.

For example: suppose I input that the principal is 100 dollars, and that the interest rates are 3.4%, 4.2%, 5.1%, 2.4% and 3.6% for years 0-2, 2-4, 4-6, 6-8, and 8-10, respectively. Then the value at year 2 (to four decimal places) will be

$$100 \cdot e^{(0.034)(2)} = 107.0365;$$

the value at year 4 will be

$$107.0365 \cdot e^{(0.042)(2)} = 116.4160;$$

the value at year 6 will be

$$116.4160 \cdot e^{(0.051)(2)} = 128.9171,$$

etc.; the final value at year 10 would be 145.3537, and that is what your program should output.

```
Enter initial investment: 100
Enter first interest rate (in percent): 3.4
Enter second interest rate (in percent): 4.2
Enter third interest rate (in percent): 5.1
Enter fourth interest rate (in percent): 2.4
Enter fifth interest rate (in percent): 3.6
The final value is $145.3537
```

(The numbers after each `:`, like `100` and `3.4` and `4.2`, 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 five 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 investment.
- ask the user five times to enter an interest rate. The program should accept the interest rates input as **percents** (but input without the percent sign – so "3.5%" will be input to the program as just `3.5`).
- clearly display the **final** account balance. Please show the value with a dollar sign, although you are not required to round to two decimal places. Note: there is no penalty for showing additional values.

***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 asks the user to input an eight-bit binary number (i.e., a string composed of 8 digits, all of which are either `0` or `1`), and prints its base-10 equivalent.

For example, a sample run of the program might look like this:

```
Enter an eight binary number: 10001110
The binary number 10001110 is the same as the decimal number 142.
```

or this:

```
Enter an eight binary number: 00110011
The binary number 00110011 is the same as the decimal number 51.
```

In the above, the strings after the colons (`10001110` and `00110011`) are user input; everything else is generated by the program.

Hint: remember that user input is interpreted as a *string*!

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

- ask for the user to input an eight-bit binary number (eight digits, all of which are `0` or `1`).
- assuming that the user enters a valid eight-bit binary number, print out the base-10 equivalent.
- 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 an integer 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 integer between 0 and 255: 24
This number is equivalent to the binary number 00011000
```

---

4) parks.py

The "center" of Central Park is located at latitude 40.782800 degrees, longitude -73.965269 degrees.

The "center" of Prospect Park is located at latitude 40.660217 degrees, longitude -73.968948 degrees.

The "center" of Flushing Meadows Corona park is located at latitude 40.739694 degrees, longitude -73.840793 degrees.

(By "center" I mean "points in the middle of the park that I randomly chose.")

Write a program that asks for a latitude and longitude from the user, and prints out which park is the closest to that point, by computing the distances from that point to the three given ones. Don't worry about the very unlikely possibility of a tie.

We assume that the point entered is in or close to New York City, so that we can treat the Earth as essentially flat. Here is a scheme for calculating distances under these assumptions, expressed in terms of kilometers. To calculate the distance between two points:

1. Calculate Δ_{Lat}, the difference between the latitudes. Multiply by 111.048 to convert the degree difference to kilometers.
2. Calculate Δ_{Long}, the difference between the longitudes. Multiply by 84.515 to convert the degree difference to kilometers.
3. The distance between the points is then found $\sqrt{(\Delta_{Lat})^2 + (\Delta_{Long})^2}$.

The strategy is simple: calculate three distances, and then use some `if` statements – or `if-elif`, or nested `if`, whatever you find to be most appropriate – to determine which distance is smallest, and print out that park.

A sample run should look like:

```
Enter Latitude:  40.743

Enter Longitude:  -73.999

Central Park is the closest park, with a distance of 5.259349843061248 km.
```

Here, `40.743` and `-73.999` are entered by the user; everything else is printed out by the program. Be sure to test your program with other points, too!

Specifications: your program must

- ask for and accept the latitude and longitude of a point within New York City.
- display the name of the park corresponding to the closest location, as calculated by the scheme described above.

***Challenge (optional)**: instead of using the "Flat Earth" approximation, use spherical geometry to calculate the distance (this will require you to use an approximation for the radius of the Earth, for which you can use 6371 km).*

---

5) monopoly.py

In the game of Monopoly, turns work as follows:

- Two 6-sided dice are rolled. The numbers shown by the dice are added, and the player moves that many spaces.
- If the two numbers rolled are different, then the turn ends; but if the two numbers happen to be the same (called a "double"), the player gets to roll again, and move his or her piece an additional number of spaces.
- If the second roll isn't a double, the turn ends; but the second roll is also a double, the player gets to roll one more time.
- If the THIRD roll is also a double, the player goes to jail; otherwise, the player moves the given number of spaces, and the turn ends.

Write a Monopoly roll simulator. It should take **no user input**. Instead, it will choose random numbers between 1 and 6; display them; and repeat as necessary. At the end, the program should display either the total number of spaces moved, or it should print `GO TO JAIL!`. The program should produce different output each time it is run.

A typical sample run might look like:

```
First roll: 5 3
You move 8 spaces.
```

But every once in a while, you should see an output that looks

```
First roll: 1 1
Second roll: 5 1
You move 8 spaces.
```

or

```
First roll: 4 4
Second roll: 3 3
Third roll: 6 1
You move 21 spaces.
```

And rarely, but sometimes, you should see an output like

```
First roll: 2 2
Second roll: 3 3
Third roll: 5 5
GO TO JAIL!
```

Specifications: your program must

- accept NO user input.
- use `random` to generate two random numbers between 1 and 6, and display those numbers.
- repeat the process up to three times if there are doubles, as described above.
- print out either the total sum of the rolls, or `GO TO JAIL!` – the latter if there are three consecutive doubles.

***Challenge (optional)**: write this program WITHOUT using nested `if`s (that is, no `if` statements inside blocks.*