MTH 3300
Fall 2018
E. Fink
September 17, 2018

Homework 3

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) quadratic.py

Write a program that prompts the user to input the three coefficients a, b, and c of a quadratic equation $ax^2 + bx + c = 0$. The program should display the solutions of this equation, in the following manner:

1. If the equation has one solution, display `ONE SOLUTION:`, followed by the solution, displayed with *4 digits printed out after the decimal place.*
2. If the equation has two real solutions, display `TWO REAL SOLUTIONS:`, followed by the two solutions, each displayed with *4 digits printed out after the decimal place.*
3. If the equation has solutions that are not real numbers, display `COMPLEX SOLUTIONS:`, followed by two solutions displayed in the form $a + bi$, where each a and b should be displayed with *4 digits printed out after the decimal place.*

For example, a run might look like

```
Enter x^2 coefficient: 1
Enter x^1 coefficient: -2
Enter x^0 coefficient: 1
ONE SOLUTION: x = 1.0000
```

or

```
Enter x^2 coefficient: 3
Enter x^1 coefficient: 5
Enter x^0 coefficient: 1
TWO REAL SOLUTIONS: x = -0.2324 and x = -1.4343
```

or

```
Enter x^2 coefficient: 2.1
Enter x^1 coefficient: 4
Enter x^0 coefficient: 10
COMPLEX SOLUTIONS: x = -0.9524 - 1.9634i and x = -0.9524 + 1.9634i
```

In the last case, note that the letter that is printed out to represent the square root of -1 is the letter `i`, **not the letter** `j`! Python has some functions that produce complex values, but when you print these values, they will display the letter `j` to represent $\sqrt{-1}$. I don't want that – so, instead, your program should calculate the imaginary coefficient, and then include code which prints out the string `''i''`.

Specifications: your program must

- ask for and accept coefficients from the user.
- assuming that the user enters a non-zero leading coefficient, display the *type* of solutions as above.
- display all solutions with exactly 4 digits printed after each decimal (for complex solutions, 4 digits after the decimal for both real and imaginary parts).
- use the letter `i` to represent $\sqrt{-1}$ in output, not the letter `j`.

***Challenge**: write the program so that it gives correct behavior for any real numbers input, even if the leading coefficient is 0. Make sure to think about **every** case!*

---

2) aubergine.py

You are working for a ride-sharing company named Aubergine. You are in charge of making the software which, once someone requests a ride, selects the car that will be sent to pick that person up.

The head office of Aubergine is located at latitude 40.740230 degrees, longitude -73.983766 degrees. As a first step towards getting your software up and running, write a program which takes as input information about two cars, and outputs a decision about which car (if any) is better to pick up someone from Aubergine headquarters.

The program should ask the user to enter the current latitude and longitude for two cars, as well as whether or not they are occupied (the user will enter the answers as either `y` or `n`, lowercase, with no spaces). The program should then calculate the distance from each of the cars to Aubergine headquarters, using the follow process to calculate the approximate distance between two locations in kilometers:

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 locations is then found $\sqrt{(\Delta_{Lat})^2 + (\Delta_{Long})^2}$.

(We're assuming that the point entered is in or close to New York City, so that we can treat the Earth as essentially flat.)

The program should then decide whether the first car or the second car – or neither! – should be sent, according to the following rules: *a car should be sent only if it is unoccupied; a car should be sent only if it is less than 10 kilometers away; if more than one car fits these criteria, the closer car should be sent.* The program should then print out the two distances, as well as the final decision about which car should be sent.

A sample run should look like:

```
Enter Latitude of Car 1: 40.686
Enter Longitude of Car 1: -73.979
Car 1 occupied (y/n): n

Enter Latitude of Car 2: 40.768
Enter Longitude of Car 2: -73.972
Car 2 occupied (y/n): y

Car 1 distance = 6.035588867293896
Car 2 distance = 3.240166507609296
Car 1 should pick you up
```

Here, the six bits following the colons (`40.686`, `-73.979`, `n`, `40.768`, `-73.972`, `y`) are entered by the user; everything else is printed out by the program. Be sure to test your program with other combinations, too!

Specifications: your program must

- ask for and accept a latitude, a longitude, and whether or not the car is occupied for two different cars (the latter should be entered as a single lowercase letter with no spaces; you may assume that the user complies).
- print out the distances of each car from Aubergine headquarters at latitude 40.740230°, longitude −73.983766°.
- display a decision about which car should pick you up, based on the criteria that a car should be sent only if it is unoccupied; a car should be sent only if it is less than 10 kilometers away; if more than one car fits these criteria, the closer car should be sent.

***Challenge**: 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). Your distances should be slightly different than mine.*

---

3) paint.py

I have a number line. Robbie Red and Bert Blue both paint segments of the line, each starting from one point, and ending at a different point.

Write a program that asks the user to enter Robbie Red's start and end points, and Bert Blue's start and end points, and outputs the total length of the number line that gets painted. Be careful: part of the number line may get painted twice, but you should only count the length once.

For example, here is a sample run:

```
Enter Robbie Red's starting point: 0.5
Enter Robbie Red's ending point: 2.5
```

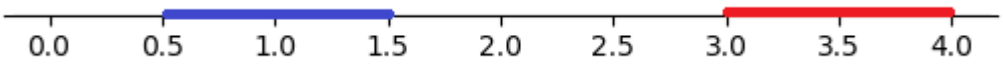
```
Enter Bert Blue's starting point: 1.5
Enter Bert Blue's ending point: 3.5
The total distance painted is 3.0
```

Here, `0.5`, `2.5`, `1.5` and `3.5` are entered by the user, and everything else is produced by the program. The answer is 3.0 since the painted part of the number line runs from 0.5 to 3.5, and 3.5 - 0.5 = 3.0. The red and blue portions overlap, but the overlap is only counted once.

Here's another sample run:

```
Enter Robbie Red's starting point: 3.0
Enter Robbie Red's ending point: 4.0
Enter Bert Blue's starting point: 0.5
Enter Bert Blue's ending point: 1.5
The total distance painted is 2.0
```

In this example, the red and blue portions don't overlap, so you simply add the two lengths: $(4-3)+(1.5-0.5)=2$. You may assume that the user enters the starting point and the ending point for Robbie Red in the correct order (i.e. the starting point is less than or equal to the ending point), and that the same goes for Bert Blue.



Specifications: your program must

- ask the user to input the starting point and ending point for Robbie Red, and the starting point and ending point for Bert Blue. You may assume that each starting point is less than or equal to the corresponding ending point.
- print out the total length of the part of the number line that gets painted when Robbie and Bert each paint from their starting points to their ending points, making sure that overlaps only get counted once.

***Challenge**: now, include Greta Green! That is, modify the program so that it allows THREE segments of the number line to be input. Again, the output should be the total painted length, with no segment counted more than once.*