MTH 3300
Spring 2019
E. Fink
February 12, 2019

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 last case*** *if you want the full 10 points!*

---

2) howl.py

You are working for a software company called Howl, which is trying to build an app which recommends restaurants. The head office of Howl 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 some information about a restaurant, and then prints out the distance from the Howl head office to the restaurant, along with a decision about whether you should try eating at that restaurant.

The program should ask the user to enter the latitude and longitude for the restaurant, as well as three user ratings for the restaurant – these should be integers, between 1 and 5, with 1 representing lowest quality and 5 representing highest quality.

The program should first calculate the distance from Howl headquarters to the restaurant, 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 restaurant 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 we should try the restaurant according to the following rules: never try a restaurant that has received any ratings equal to 1, that has an average rating of less than 2.5, or that is greater than 4 kilometers away; for restaurants that are between 2 and 4 kilometers away, only try the restaurant if the average rating is ≥ 3.5 (i.e., you have higher standards for restaurants that are moderately far away than you have for close restaurants).

A sample run should look like:

```
Enter latitude of restaurant: 40.768
Enter longitude of restaurant: -73.972
Enter first review score: 3
Enter second review score: 3
Enter third review score: 4
Distance = 3.240166507609296
Don't try
```

Here, the five bits following the colons (`40.768`, `-73.972`, `3`, `4`, `4`) are entered by the user; everything else is printed out by the program. This example prints `Don't try` because, while there are no scores that equal 1, the restaurant is moderately far away (between 2 and 4 kilometers), and the average review is about 3.33, which is less than 3.5. Another sample run might look like:

```
Enter latitude of restaurant: 40.742
Enter longitude of restaurant: -73.983
Enter first review score: 3
Enter second review score: 3
Enter third review score: 4
Distance = 0.2069418381770326
Try
```

Since this restaurant is so close (less than 2 kilometers away), the average score of 3.33 is acceptable – it's ≥ 2.5, and none of the reviews are 1's. Be sure to test your program with other combinations, too!

Specifications: your program must

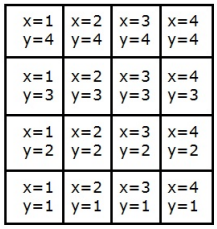
- ask for and accept a latitude, a longitude, and three different review scores (the review scores are to be entered as integers between 1 and 5; you may assume that the user complies).
- print out the distance (in kilometers) of the restaurant from Howl headquarters at latitude 40.740230°, longitude −73.983766°.
- display a decision about whether you should try the restuarant, based on the criteria that: you never try a restaurant that has received any ratings equal to 1, that has an average rating of < 2.5, or that is > 4 kilometers away; for restaurants whose distance is between 2 and 4 kilometers away (inclusive of 2 and 4), only try the restaurant if the average rating is ≥ 3.5.

***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.*

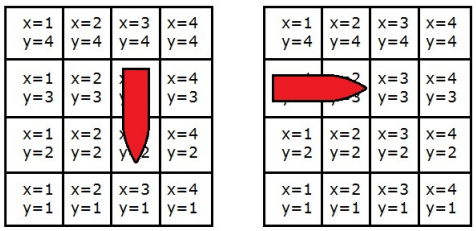
---

3) battleship.py

In a version of the game Battleship, you place "ship" pieces onto a grid. The grid is labelled as follows:



A ship piece should take up two adjacent squares (*adjacent* meaning horizontal or vertical, not diagonal) that lie on the grid. For example, here are two valid placements:



Write a program that allows the user to input the x- and y-coordinates of two squares, and then prints out whether or not they form a valid ship – meaning that both squares lie on the grid, and are adjacent to one another horizontally or vertically. If one of those conditions is not met, the program should print out a message saying which condition is wrong. (If *both* conditions are not met, the program only needs to output a single message, which can be either one, your choice.)

For example, here is a sample run:

```
Enter x-coordinate of square one: 3
Enter y-coordinate of square one: 2
Enter x-coordinate of square one: 3
Enter y-coordinate of square one: 3
Valid
```

– here, the numbers are input by the user, and everything else is produced by the program; this is `Valid` because it corresponds to the first placement shown in the figure above. Other sample runs might look like:

```
Enter x-coordinate of square one: 3
Enter y-coordinate of square one: -1
Enter x-coordinate of square one: 5
Enter y-coordinate of square one: 2
Square(s) not on grid
```

(here, both $(3, -1)$ *and* $(5, 2)$ happen to be outside the grid), or

```
Enter x-coordinate of square one: 3
Enter y-coordinate of square one: 2
Enter x-coordinate of square one: 1
Enter y-coordinate of square one: 2
Non-adjacent squares
```

(since $(3, 2)$ and $(1, 2)$ are not horizontally or vertically adjacent).

Specifications: your program must

- ask the user to input the x- and y-coordinates for two squares. You may assume that the user enters integers.
- print out a message saying that the placement is valid if both squares are on the grid (x and y coordinates are each integers between 1 and 4, inclusive), and the two squares are horizontally or vertically adjacent (i.e., the second square is either one box above, one box below, one box to the left of, or one box to the right of the other square).
- print out a message if the user inputs an invalid ship, giving a specific reason – either that the coordinates of at least one square are off the grid, or that the squares provided are not horizontally or vertically adjacent. (If both conditions are not met, you only need to print out one message.)

***Challenge**: allow the user to input THREE ships (that would be 12 input statements). Have your program print out a message, confirming whether or not all the placements are valid AND none of the ships overlap. Think carefully about how you would write this so that the code doesn't become a mess!*