MTH 3300
Fall 2018
E. Fink
November 29, 2018

Homework 10

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 ONE OF THE PROBLEMS!!!!!!!! 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) paint2.py

Recall the paint problem from Homework 3: the user inputs two intervals on the number line, and your job was to determine the total length of the number line that is "painted" by those two intervals – keeping in mind that they may intersect, or that one may subsume another.

I have a file called `intervals.txt`, where each row contains an interval (in correct order). Find the total painted length of all the intervals, using a recursive function named `recursive_paint`.

Here's the strategy: `recursive_paint` will receive a 2D list, which will consist of lists containing two floats in order – for example, something like `x = [[1,4], [2.4, 3.8], [5,6], [3,8]]`). The function will take the *last* of the elements in the list, and then check if it overlaps with (or completely contains, or is contained in) any of the previous intervals. If it DOES: then combine those two intervals into one, and find the length of the new shorter list. If it does NOT: simply add the length of the last interval to the total painted length of all the other intervals.

For example: if the list that is input is `[[1,3],[2,5],[0,4],[-12,-8]]`, the function will just add 4 to the painted length of `[[1,3],[2,5],[0,4]]`, since $-8-(-12)=4$. If the input is `[[1,4],[12,14],[7,9],[8,13]]`, the function will combine `[8,13]` with `[12,14]` (which is the first entry that it intersects) to form the combined interval `[8,14]`, and then the function will return the painted length of `[[1,4],[8,14],[7,9]]`.

When reading the file, be sure to pay attention to the `[`, `,` and `]` characters on each line. Slicing might help in turning each line from a string to a list of floats. With the file I provided, the total painted length should be about 56.1373.

Specifications: your program must

- read a file named `intervals.txt`, which contains a number of intervals in the format of my example.
- compute the total painted length of the contained intervals (as described in Homework 3 – alternatively, this can be described as finding the length of the union of the intervals).
- utilize a recursive function.

---

2) movies.py

I have a (gently altered) database of film permits issued by the city of New York contained `film_permits.db`. The data is contained in the table `Permits` in that database – the column names are the same as those contained in the `check.csv` (which, as the name suggests, I've included just so that you have the option of easily checking parts of your program), and each column is typed as `TEXT`.

Write a program that allows the user to enter a `Month` and a `Year`. The program should access all the rows from the table which have the given `Month` and `Year`, which also have their `EventType` as `Shooting Permit`. Among these rows, the program should print out the *most frequently occurring zip code(s)*. Be aware that some rows will have multiple entries in their `ZipCode` column: for these rows, each of the different `ZipCode`'s should be counted once. I'd recommend using a dictionary to keep track of the counts.

Also pay attention to the fact that when you fetch SQLite queries, each row reported will be given as a list (or, more accurately, a tuple), *even if you only queried for one column.*

Specifications: your program must

- allow the user to input a month and a year.
- access all rows in the table `Permits` in the SQLite database `film_permits.db` with the given `Month`, the given `Year`, with the `EventType` field equal to `Shooting Permit`.

- among these entry, report the zip code(s) that appear(s) most frequently in the `ZipCode` column.

Challenge: do both problems!