CIS 41A Lab 8: Regular expression and data structures

This lab assignment introduces regular expression, but it is also a review of concepts that we've learned this quarter. There are only a few requirements so it's up to you to come up with an efficient design by using the correct data structures.

Overview:  
The program will read data from a text file which contains results of a bicycling race, then help the user look up information of the race.

Input text file:  
The input text file is at lab8input.txt (source: https://www.webscorer.com/resources/templateresults). The data is from a bicycle race in Washington state, and most lines of the file contain information fields for one bicyclist: **id number**, **name**, **team name**, **location**, **racing distance**, **racer type**, and **race times**. Some fields are optional.

Test driver:  
Download the main lab8.py to be the test driver for your part of the program: a class called raceAnalyzer.  
Your raceAnalyzer class should work with lab8.py, without any change to lab8.py.

Your part:  
Here are the requirements for the **raceAnalyzer**:

1. Use regular expression to extract the **name**, **location**, **racing distance**, **racer type**, and **finish time** (first time field).

* Hint 1: you should be able to write **1 regex** to extract all the required fields
* Hint 2: take some sample lines from the lab8input.txt file and create a smaller file with about 10 lines to use for testing your regex. Choose the sample lines carefully so you get all different types of lines. Don't just copy the first 10 lines or last 10 lines of lab8input.txt
* Hint 3: build your regex slowly, 1 field at a time. Start with a regex that extracts just the name field, test to see that it works. Then add to the regex to extract the location, then test to see that you can get both fields, before continue to add to the regex for the next field.

2. Handle all user requests, which are:

* search for a racer by name (case insensitive):
  + list the matching name, racing distance, and finish time
* search by race type:
  + list all racer names (in alphabetical order) within a type. There are 2 main types and 2 sub-types. The 2 types of races are: 50 mile or 100 mile. Within the distance, the 2 sub-types (for age) are: open or masters. Print the number of racers of each type. See sample output.
* search for location:
  + list the count of racers from each state.
  + The state should be the 2 letter abbreviation, for example: CA, WA (Washington), OR (Oregon)... To keep it simple, you can assume the first 2 letters of the state name is always the correct abbreviation. (Challenge: can you convert the state name in the file into the 2 letter abbreviation in one line of code?)

Hint for storing data and for searching: the title() method of the str class is very useful to make all text uniform for case insensitive searching and data storing.   
At the Python shell, type: "abc def".title() and "ABCd eFg".title() to see what title() does.

3. There should be at least 5 methods in the raceAnalyzer class, for reading in data and for each of the searches.

4. Before you dive into the coding phase, it's a good idea to think about the design for your raceAnalyzer class.

What data structure is best to store the data so that the searches can be fast?

Perhaps you need to store the data into more than one data structure?

On one hand, you don't want to create duplicate data structures, but on the other hand, the code can be simpler and faster if data is created once and 'cached' in a data structure rather than re-created with every data look up. (This concept is called memoization and Python has support for it - a topic in CIS 41B)

When you've decided on which data structure(s) to use, then for each data structure that you create in the code, add a comment next to it to explain what it is. For example: dictionary with key = name, value = time. Or: list of lists (distances).

*"I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships." -- Linus Torvalds* [*https://lwn.net/Articles/193245/*](https://lwn.net/Articles/193245/)

5. When done testing, submit your raceAnalyzer.py file.

Sample program output (user input in orange):

Total: 219 racers

Choose from:

n. search for racer by name

t. print all result by race type

l. print count of racers by location

q. quit

Your choice: l

WA: 179 # state abbreviation are all uppercase

OR: 31

BC: 3

CA: 3

ID: 1

MA: 1

UT: 1

Choose from:

n. search for racer by name

t. print all result by race type

l. print count of racers by location

q. quit

Your choice: t

50 Mile Distance:

Open

Adams, Mike # names are in alphabetical order

Adsero, Nick

Anderholm, Ivan

... <cut to save room> ...

Sherrick, Bob

Shulock, Adrian

Smith, Robin

... <cut to save room> ...

Woerner, Jeff

Wright, Andrew

Zeigler, Andrew

137 racers in the 50 mile Open race # print the count for each type

Masters

Almazora, Lito

Benish, Frank

Brady, Steven

... <cut to save room> ...

Winger, Terry

Zars, Hugh

52 racers in the 50 mile Masters race

100 Mile Distance:

Open

Atkinson, Monilee

Belair, Lisa

... <cut to save room> ...

White, Shari

20 racers in the 100 mile Open race

Masters

Albright, Jim

... <cut to save room> ...

Russo, Ted

Sheppard, Michael

Trombley, Robert

10 racers in the 100 mile Masters race

Choose from:

n. search for racer by name

t. print all result by race type

l. print count of racers by location

q. quit

Your choice: n

Enter a racer full name: dale, pat

Name: Dale, Pat

Distance: 50 miles

Time: 4:32:23.9

Choose from:

n. search for racer by name

t. print all result by race type

l. print count of racers by location

q. quit

Your choice: n

Enter a racer full name: rugh, david

Name: Rugh, David

Distance: 100 miles

Time: DNF # DNF is for Did Not Finish, it's considered a "time" in races

Choose from:

n. search for racer by name

t. print all result by race type

l. print count of racers by location

q. quit

Your choice: n

Enter a racer full name: racer # name not found

No racer by the name Racer

Enter a racer full name: n

No racer by the name N

Enter a racer full name: albright, jim

Name: Albright, Jim

Distance: 100 miles

Time: 10:35:44.0

Choose from:

n. search for racer by name

t. print all result by race type

l. print count of racers by location

q. quit

Your choice: q