CS 224 Introduction to Python | Spring 2022

Closed Lab 01

University of Wisconsin – La Crosse | *February 4*

Lists and Files

Description: Along with this document, you downloaded two files: `data.csv` and `lab01.py`. The former contains data from an experiment involving the Iterated Prisoner's Dilemma game; the latter is the Python source code file that you will edit.

The Iterated Prisoner's Dilemma is a two player game in which criminal accomplices have been captured by the police. They are being held, and questioned, separately (you can almost hear the intro voiceover from *Law & Order*). Each has two choices: confess and implicate their accomplice (*defect*), or remain silent (*cooperate*). The penalty that each receives depends not only on their own decision but on that of the other player as well.

`data.csv` contains results from an IPD competition in which a large number of players played a round-robin tournament in which each match consisted of 150 rounds. Each row in the file contains the information for one player. Note that the number of data elements in a row is not uniform across players.

In `lab01.py`, I have included code that reads the contents of `data.csv` and stores them in a Python list named `data_lines`. Each element in the list represents a row from the file. You will add code to manipulate the data in several ways.

Part 1: Sort the data by player type, the last entry in each line. Player type values are not unique. You need not worry about the order of players within a player type. Print the contents of the list to verify that it is sorted. Now sort the list by average score.

Reading from the end of a row, this is the third real-valued entry. Once again, print the contents of the list to verify that it is correctly sorted.

Part 2: This part again uses `data_lines` as a starting point. You must create a new list that contains the count of the players of each player type. The types are numbered 0 through 49. Determine if any of the types occur zero times. If so, report which types.

Part 3: Sort `data_lines` by player type. Now, within each player type, sort the rows by average score. The result should be all players with type 0 at top of file, in order by average score, followed by all players of type 1, in order by average score, and so on. What challenges do you encounter with this problem?