CS 1803

Pair Homework 4 – Greedy Scheduler (Part I)

Due: Wednesday, February 23rd, before 6 PM
Out of 100 points

Files to submit: 1. HW4.py

<u>This is a PAIR PROGRAMMING Assignment: Work with your partner!</u>
For pair programming assignments, you and your partner should turn in identical assignments. List both partners names at the top. Your submission must not be substantially similar to another pairs' submission. Collaboration at a reasonable level will not result in substantially similar code. Students may only collaborate with fellow students currently taking CS 1803, the TA's and the lecturer. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. You should not exchange code or write code for others.

For Help:
- TA Helpdesk – Schedule posted on class website.
- Email TA's or use T-Square Forums

Notes:

- **Don't forget to include the required comments and collaboration statement (as outlined on the course syllabus).**
- ***Do not wait until the last minute*** *to do this assignment in case you run into* problems.
- **Read the entire specifications document before starting this assignment.**

Premise

In this assignment, you will be tasked with reading in data about meeting times for various courses. Once you read in this data, you will be required to insert the data into the given data structure which consists of a Dictionary, lists, and tuples. After inserting all of the data into the data structure, you will be required to print out the structure in a readable format. Details about the file you will be reading, the data structure you will be creating, and the output format will be covered later in this specification. In this assignment, you will be required to write the following four functions:

1. insertIntoDataStruct()
2. parseLine()
3. readCSVFile()
4. printDataStruct()

File Format Information

For this assignment, you will be given a comma separated value (CSV) file. This file will contain the following information: The name of a course, followed by pairs of start times and end times. All of this data is separated by commas, as is standard in a CSV file. Take the following example line:

```
CS 1803,1505,1555,1405,1455
```

CS 1803 is the course name. After the course name, there will be some number of start time and end time pairings, listed in 24 hour format with no punctuation (1:30 PM would be listed as 1330 in the file). In the example above, there are two times this course meets: 1505 – 1555 and 1405 – 1455. You do not need to convert the times into any other format. You may safely assume that every start time will have an end time and that all times are properly formatted (i.e. there are no bad times such as 1660 in the file, or times which contain anything other than digits). You should, however, ignore lines which contain only whitespace characters (The strip() function may be useful again.).

NOTE: Do not edit the CSV file in a program like Excel. If you do, it will throw in garbage which will cause your program to parse the file incorrectly. Use a program like Notepad to edit the file.

Data Structure Information

The primary data structure for this assignment will be a dictionary named **courses**. Remember that dictionaries contain a key which is how you look up information in the dictionary, and each key has an associated value with it. In this particular dictionary, the key will be the name of the course and the value will be a list which contains the course time pairs. The course time pairs are stored as a tuple where the first element in the tuple is the start time (represented as an **integer**) and the second element in the tuple is the end time (also represented as an **integer**).

An example of this structure with actual data looks something like this:

“CS 1803” → [(1505,1555),(1405,1455)]
“CS 4400” → [(1305,1335),(1405,1435),(1505, 1535)]

Note that the list can have any number of pairings in it, but each key will have at least one pairing in its value list.

You may assume that the data structure will be emptied by the user before reading a new file.

Function Name: **insertIntoDataStruct**

Parameters:

string – A string which contains the name of the course.
string – A string which contains the start time of the course.
string – A string which contains the end time of the course.
Dictionary – The data structure to add the course time to.

Return Value:

Dictionary - The dictionary with the new course time pairing added to it.

Description:
Write a function that will accept four parameters: the first parameter is a string which is the name of the course; this will serve as the key when you actually add the tuple you will create to the list of times the course meets. The second parameter is a string representing the start time of the class, and the third parameter is a string representing the end time of the class. The last parameter is the dictionary which you will be adding the course time to. Using the two times you are given, you must first construct a tuple containing the start and end times of the course, in that order (see the Data Structure Information above). Then you must add the tuple you just created to the list of tuples which represents the times when the given course meets. You will need to create this list in the event that you are not adding a time pairing to a course which is already in the dictionary (courses). Otherwise, you should add the tuple to the existing list of times for that course. Do not exclude duplicate time pairs from the dictionary.

Function Name: **parseLine**

Parameters:

string – A string which contains the contents of one line from the file
Dictionary – the data structure which to add the course-time pairings to.

Return Value:

Dictionary – the data structure containing the course-time pairings.

Description:
Write a function that will accept two parameters. The first is a string which contains the contents of one line from the file you are reading in the function readCSVFile(). The second is the dictionary to which you will be adding the times from the line to. The parseLine() function will then separate out the parts of the file, then make the appropriate calls to insertIntoDataStruct() to put the information it has extracted from the line of the file into the data structure. You must call insertIntoDataStruct() to put the data into the structure; do not insert the parsed data directly into the data structure inside of parseLine().

Function Name: **readCSVFile**

Parameters:

string – A string which contains the name of the file to read in
Dictionary – An empty dictionary to add the courses to.

Return Value:

Dictionary – the data structure containing the course-time pairings.

Description:
Write a function that will accept two parameters. The first is a a string which contains the name of the CSV file you wish to parse (for example, "coursetimes.csv"). The second is the dictionary into which you will add all of your courses and their respective times from the file. Your function will then read in this file and call parseLine() on each line to parse the information contained within that line. You must call parseLine() inside of readCSVFile() to parse each line of the file; the only thing that this function should do is read in the contents of the file and pass each line to parseLine(). In the event that the user provides an invalid file name, you should print a message such as "Invalid file name." and force the user to enter a valid file name. Remember to close the file when you're done reading in the information. You may not use the CSV reader module; you must implement CSV reading yourself.

Function Name: **printDataStruct**

Parameters:

Dictionary – The data structure containing all the course-time pairings

Return Value:

none

Description:
This function will display the contents of your data structure in a format which is more easily readable than Python's built-in dictionary printing format. The format should look like the following:

```
Course Name:
        Start time – End time
        Start time – End time
        …
        Start time – End time

Course Name:
        Start time – End time
        Start time – End time
        …
        Start time – End time
```

The order in which the courses and their times are printed is unimportant so long as each start time and end time pair is printed under the proper course. You do not need to convert the times to 12 hour format or add colons between the hours and minutes; simply print out the integer values as they are stored in your data structure.

Testing Your Code (Test Cases)

insertIntoDataStruct:

Assuming at the beginning of this test the dictionary is empty and the dictionary is named ***courses****.*

```
courses = insertIntoDataStruct(“CS 1803”, “1505”, “1555”, courses)
courses = insertIntoDataStruct(“CS 1803”, “1405”, “1455”, courses)
courses = insertIntoDataStruct(“CS 1803”, “1305”, “1355”, courses)
courses = insertIntoDataStruct(“CS 4400”, “1405”, “1455”, courses)
courses = insertIntoDataStruct(“CS 4400”, “1605”, “1655”, courses)
print(courses)
```

{'CS 1803': [(1505, 1555), (1405, 1455), (1305, 1355)], 'CS 4400': [(1405, 1455), (1605, 1655)]}

Note that your results may be in a slightly different order as there is no defined order for dictionaries. Just ensure that all of the data has been successfully placed into the dictionary.

parseLine:

Note that the functionality of the parseLine() function depends heavily on insertIntoDataStruct() working correctly; it is advisable that you have insertIntoDataStruct() coded before testing parseLine().

Assuming at the beginning of this test the dictionary is empty and the dictionary is named ***courses****.*

```
courses = parseLine(“CS 1803,1505,1555,1405,1455,1305,1355”, courses)
print(courses)
```

```
{'CS 1803': [(1505, 1555), (1405, 1455), (1305, 1355)]}
```

```
courses = parseLine(“CS 4400,1405,1455,1605,1655”, courses)
print(courses)
```

{'CS 1803': [(1505, 1555), (1405, 1455), (1305, 1355)], 'CS 4400': [(1405, 1455), (1605, 1655)]}

readCSVFile:

Note that the functionality of the readCSVFile() function depends heavily on parseLine() working correctly; it is advisable that you have parseLine() coded before testing readCSVFile().

Assuming at the beginning of this test the dictionary is empty and the dictionary is named ***courses****. This test uses the included courseTimes.csv file.*

```
courses = {}
courses = readCSVFile("courseTimes.csv", courses)
print(courses)
```

{'CS 1803': [(1505, 1555), (1405, 1455), (1305, 1355)], 'CS 4400': [(1405, 1455), (1605, 1655)], 'MATH 2402': [(1405, 1525)]}

printDataStruct:

Assuming at the beginning of this test the dictionary is empty.

```
courses = insertIntoDataStruct("CS 1803", "1505", "1555", courses)
courses = insertIntoDataStruct("CS 1803", "1405", "1455", courses)
courses = insertIntoDataStruct("CS 1803", "1305", "1355", courses)
courses = insertIntoDataStruct("CS 4400", "1405", "1455", courses)
courses = insertIntoDataStruct("CS 4400", "1605", "1655", courses)
printDataStruct(courses)
```

```
CS 1803:
        1505 – 1555
        1405 – 1455
        1305 – 1355

CS 4400:
        1405 – 1455
        1605 – 1655
```

It is strongly recommended you also write your own additional test cases which include blank (whitespace) lines in the CSV file, as well as a longer set of inputs.

Grading:

You will earn points as follows for each function that works correctly according to the specifications.

insertIntoDataStruct()		**30**
Converts start and end times to integers	5	
Properly handles input when course already exists	13	
Properly handles input when course does not exist	12	
parseLine()		**20**
Properly handles the course name	5	
Properly handles all time pairs in the line	10	
Properly calls insertIntoDataStruct()	5	
readCSVFile()		**30**
Properly handles invalid file name	6	
Properly handles multiline files	6	
Properly ignores whitespace lines	6	
Properly passes lines in file to parseLine()	6	
Closes the file before exiting the function	6	
printDataStruct()		**20**
Output properly formatted	10	
Output contains all entries in dictionary	10	