**CPSC111BV - Introduction to Computation Winter 2018**

**Final Project – 2-D lists & Magic Squares**

**Name and student #: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Name and student #: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

***Project MARK: \_\_\_\_\_ / 100* Due Date: Friday, April 6th (12:00 PM)**

**Submission**: Submit your project to the instructor by email at: [drahmalki@gmail.com](mailto:drahmalki@gmail.com)

Complete the following programming questions, for each question:

* Ensure you have proper program documentation
* Ensure that each method/function is documented/explained
* Submit sample output of your program (via email)
* Submit the Python source code (via email)
* Ensure that your project starts with a **Python Template**

**Do not wait until the last minute to do this project in case you run into problems.**

**Remember to comment your code, choose appropriate variable names, and use consistent spacing and indentation!**

**Note:** *You are responsible for protecting your solutions to these problems from being seen by other students,* either physically (e.g., by looking over your shoulder) or electronically.

**Final Project statement**

A magic square is a 2-D list where the sum of each row is equal to the sum of each column and is equal to the sum of each of the two diagonals. Here is an example of a 3 x 3 magic square:

4 9 2

3 5 7

8 1 6

Note that the sum of each of the rows, columns, and diagonals is 15. This square is not unique. If you flip the rows for the columns (transpose the square) you will get a magic square. If you add a constant value to each element you will get a magic square, and so on.

For this assignment, you will write a program that reads squares in from a file, determines whether they are magic, and writes the output to a file.

You may work in pairs for this assignment, but you must follow the guidelines covered in Pair Programming Guidelines.

Note: You may not use any programming construct that we have not covered in class.

File Name: MagicSquares.py

In your main() function, prompt the user to enter the name of the input file and the name of the output file. If the names are the same, write a message to that effect and quit the program. If the names are different, open the input file for reading and the output file for writing and process the data. When the program has completed processing all the squares, write a message to the console that the output has been written to output file. A sample session would look like this (user input in green):

Enter name of input file: squares.txt

Enter name of output file: result.txt

The output has been written to result.txt

For this program, write a function isMagic() that determines if a 2-D list forms a magic square. The function should be general enough to accept magic squares of any size greater than or equal to 3. The function should accept a 2-D list as a parameter and return a Boolean value that indicates whether the square is magic or not. This function must call at least four (04) other functions.

**Input:**The input for this program is a file. The first line of the file is a number *n* denoting the number of squares that need to be processed. It is followed by the data for each of the *n* squares. There is a blank line separating data for each square. For any given square the first line will give the number of rows (or the columns) followed by the rows of the square---one row per line. Here is an example:

2

3

4 1 2

3 5 7

8 9 6

4

16 3 2 13

5 10 11 8

9 6 7 12

4 15 14 1

The input file name is of your own choosing. But you can use this sample file [squares.txt](http://www.cs.utexas.edu/~ans/classes/cs303e/assignments/squares.txt)to test your code. You may assume that the input file is in the correct format.

**Output:**The program will output to a file. The name that you choose for your output file should be different from the name of the input file. The format of the output file should be similar to the input file. You will write *valid* or *invalid* next to the size of each square. For the above input file, the output file should look like this:

2

3 invalid

4 1 2

3 5 7

8 9 6

4 valid

16 3 2 13

5 10 11 8

9 6 7 12

4 15 14 1

Test your program by running it several times with different inputs. Put your test runs **and their output** in comments at the end of your .py file

Name your file MagicSquares.py. Be certain to begin your file with the following header:

# File: --name of file--

# Description: --a description of your program--

# Final Project term: winter 2018

#

# Name: --your name--

# SID: --your Studentid--

# Course Name: --your course number--

#

# Name: --your name--

# SID: --your Studentid--

# Course Name: --your course number--

#

# Date created:

# Date last modified:

#

Did you remember to

* do this project with a partner, did you follow the pair programming guidelines?
* name the file correctly?
* put your program in a main function?
* match your output to the sample output?
* use meaningful variable names?
* use comments to document your program?
* include test cases?
* be certain your file compiles properly? (If not, it is a 0!)
* put the right header at the top (and change the header if you used pair programming)?

*Try to use good variable names and keep your code as readable as possible along with a good Program Documentation. Coding style is part of the marking scheme.*

Your program will be graded according to the following [grading criteria](http://www.cs.utexas.edu/~ans/classes/cs303e/GradingCriteria.html):

|  |  |
| --- | --- |
| **Grading Criteria for Programming Project** | |
| **A. Program Design (25%)** | |
| **Rating** | **Criteria** |
| 25 | Solution well thought out |
| 15 | Solution partially planned out |
| 05 | ad hoc solution; program "designed at the keyboard" |
| **B. Program Execution (20%)** | |
| **Rating** | **Criteria** |
| 20 | Program runs correctly |
| 12 | Program produces correct output half of the time |
| 04 | Program runs, but mostly incorrectly |
| 0 | Program does not compile or run at all |
| **C. Specification Satisfaction (25%)** | |
| **Rating** | **Criteria** |
| 25 | Program satisfies specification completely and correctly |
| 15 | Many parts of the specification not implemented |
| 05 | Program does not satisfy specification |
| **D. Coding Style (20%)** | |
| **Rating** | **Criteria** |
| 20 | Well-formatted, understandable code; appropriate use of language capabilities |
| 12 | Code hard to follow in one reading; poor use of language capabilities |
| 04 | Incomprehensive code, appropriate language capabilities not used |
| **E. Comments (10%)** | |
| **Rating** | **Criteria** |
| 10 | Concise, meaningful, well- formatted comments |
| 06 | Partial, poorly written or poorly formatted comments |
| 04 | Wordy, unnecessary, incorrect, or badly formatted comments |
| 0 | No comments at all |
| **F. Extra Credit (10%)** | |
| **Rating** | **Criteria** |
| 04 | Programs that usefully extend the requirements |
| 03 | Programs that use a particularly good algorithm |
| 03 | Programs that are particularly well written or use the capabilities of the language particularly well |

**N.B:** Next, you are going to make sure you can use the submission properly. This will be how you turn in your projects electronically. The submission procedure only accepts **zip files (NOT rar Files)**, so you are going to create a “FinalProject\_YourFirstName\_YourStudentID.zip” file containing all the programs you just wrote as well as this document with your name and student number added to it. To create a “.zip” file, find the directory where the files are saved and select all of them. If you right-click on one of the selected files, you should have the option to create an archive by choosing “Send to” then “Compressed “zipped” Folder”. Rename the created “.zip” file and give it the name FinalProject\_YourFirstName\_YourStudentID.zip.

10 marks will be deducted if:

1. the email did not have a full name and a student # in the subject field or
2. the procedure for zipped file is not followed or
3. Python template and documentation are missing or
4. the Python source code or
5. For all questions, sample outputs of your programs are not submitted or
6. This document, with your name and student number on it, is not included with the zipped file or
7. Your partners’ email is not included in the CC or BCC of the email.

**A score of zero (0) will be given to every empty email.**

**Incomplete projects may be accepted for partial credit but no late work on this project will be accepted after Friday, April 6th (12:00 PM)**