CSE 101: Introduction to Computational and Algorithmic Thinking

Homework #3-2

Fall 2018

Assignment Due: Monday, December 10, 2018, by 11:59 PM

Getting Started

This assignment contains a "bare bones" file with a name like `list-tricks.py`. This file contains one or more function and/or class *stubs* and a few tests you can try out to see if your code seems to be correct. You must fill in the missing code as directed in order to complete the assignment. **Do not, under any circumstances, change the names of any classes or functions (including those functions' parameter lists) that we provide in the starter code.** The automated grading system will be looking for exactly those class and function names (you are more than free to add your own helper functions and classes if you desire). **Please note that the test cases and data files that we give you are not necessarily the same ones we will use during grading!**

Directions

This assignment is worth a total of 24 points. The automated grading system will execute each function in your solution several times, using different input values each time. Each test that produces the correct/expected output will earn 1 or more points, adding up to the total point value available for that function. You must earn at least 18 points (75%) in order to pass (receive credit for) this assignment.

This assignment counts towards Course Outcome 3: Using Algorithms to Develop and Express Solutions to Computational Problems.

- At the top of the `list-tricks.py` file, include the following information **in full-line comments**, with each item on a separate line:
  - your full name ***AS IT APPEARS IN BLACKBOARD***
  - your Stony Brook ID #
  - your Stony Brook NetID (your Blackboard username)
  - the course number (CSE 101) and semester (Fall 2018)
  - the assignment name and number (Assignment #3-2)

⚠ Any functions that we tell you to complete (i.e., that we provide stubs for in the starter code) **MUST** use the names and parameter lists indicated in the starter code file. Submissions with different function names (or whose functions contain different parameter lists) can't be graded by our automated grading system.

⚠ Be sure to submit your final work as directed by the indicated due date and time. Late work will not be accepted for grading. Work is late if it is submitted after the due date and time.

⚠ Programs that crash will likely receive a grade of zero, so make sure you thoroughly test your work before submitting it.

List Manipulation in Python

Part I: Magic Squares (12 points)

A *magic square* is a N x N matrix of cells where each cell contains one of the first N^2 positive integers. The chief property of a magic square is that that the sum of the numbers along any row, column and the two diagonals is a constant. The following algorithm can be used to generate magic squares for odd values of N:

1. Write 1 in the middle cell of the top row.
2. Now keep writing 2, 3, 4, ... in the up-left direction. If you reach the end of the square, "come out" on its opposite side. For example, if moving to the left means that you cross over the left edge of the square,"wrap around" to the right side of the square and use that cell instead.
3. If you have to write on a used square, go down one cell from your starting position instead.

For example, here is a 3 by 3 magic square:

```
6  1  8
7  5  3
2  9  4
```

Note that, after placing the value 3, the next cell (one row up, one column to the left) is already occupied by 1. As a result, 4 is placed in the cell below the starting position (3) instead.

General update strategy: After you place a value into the magic square, use temporary variables to predict the next row and column, which are initially (current row - 1) and (current column - 1).

- If either temporary value is less than 0 (meaning that you've run off the edge of the square), reset it to n - 1.
- Check to see if the value at the new position is -1. If it isn't:
    - change the temporary row to row + 1 (moving down one row). If the next row is now $\geq$ n, set it to 0
    - set the temporary column to the current column
- Assign the temporary row and column variables to your "normal" row and column variables

Complete the `magicSquare()` method, which takes an odd, positive integer *N* as its argument and returns a filled-in magic square (a list of N N-element lists) of that size. Your function should check to make sure its argument is a positive, odd integer; if it isn't, the function should return the predefined Python value `None` instead.

Hint: Start by creating an empty list. Use a loop to insert N lists into this outer list. Each inner list should contain N elements, each of which is the value -1, indicating an empty or unused cell (this is easy to do: just create a one-element list with your desired value and multiply it by N, as in `row = [-1] * n`).

Hint: Remember that you need ***TWO*** index values to refer to a value in a nested list. The first index refers to the sublist, and the second index refers to the value's position within that sublist. For example, `square[1][3]` would refer to the fourth element (index 3) in the second sublist (index 1 in the original/enclosing list). **Be careful:** your indices will run from 0–(n-1), **NOT** 1–n!

Examples:

Function Call	Return Value
`magicSquare(-3)`	`None`
`magicSquare(5)`	`[[15, 8, 1, 24, 17], [16, 14, 7, 5, 23], [22, 20, 13, 6, 4], [3, 21, 19, 12, 10], [9, 2, 25, 18, 11]]`
`magicSquare(7)`	`[[28, 19, 10, 1, 48, 39, 30], [29, 27, 18, 9, 7, 47, 38], [37, 35, 26, 17, 8, 6, 46], [45, 36, 34, 25, 16, 14, 5], [4, 44, 42, 33, 24, 15, 13], [12, 3, 43, 41, 32, 23, 21], [20, 11, 2, 49, 40, 31, 22]]`
`magicSquare(4)`	`None`

Neatly-formatted versions of the 5x5 and 7x7 magic squares listed above are as follows:

```
15  8  1 24 17
16 14  7  5 23
22 20 13  6  4
 3 21 19 12 10
 9  2 25 18 11
```

```
28 19 10  1 48 39 30
29 27 18  9  7 47 38
37 35 26 17  8  6 46
45 36 34 25 16 14  5
 4 44 42 33 24 15 13
12  3 43 41 32 23 21
20 11  2 49 40 31 22
```

Part II: Major Verification (12 points)

The Computer Science major at Stony Brook University, like every major course of study, requires students to complete a specific set of courses. Each course must be completed with a final grade of 'C' or higher. Complete the `checkCourses()` function, which takes two arguments: a list of required course numbers (e.g., "CSE114") and a string representing the name of a data file. The data file contains a series of course numbers (the keys) and final grades (values, represented as integer). For simplicity, we will consider a final grade of 75 or higher as satisfying the "C or higher" requirement.

Your function should do the following:

1. Open the data file and create a new dictionary with its contents. Each line of the data file has two elements: a course name (a string consisting of three uppercase letters and three digits, like "CSE220") and an integer value in the range 0–100 (inclusive) representing a final grade, separated by a single space.
2. Use a loop to process the list of required courses, one element at a time. For each course, check to see if it is in the dictionary and, if so, if its value (the final grade) is greater than or equal to 75. If the course is **NOT** in the dictionary, or if its associated final grade is less than 75, add the course name to a new list variable representing requirements that have not yet been satisfied.
3. Once the entire list of required courses has been processed, the function should return one of two values: the list of unsatisfied required courses, or the Boolean value `True` if all required courses have been satisfied.

For example, given the requirements `["CSE114", "CSE214", "CSE215", "CSE216"]` and the transcript dictionary `{"CSE114":85, "CSE215":90, "CSE214":70}`, `checkCourses()` would return the list `["CSE214", "CSE216"]`, which are the remaining required courses (the student didn't earn a final grade of 75 or higher in CSE 214, and didn't take CSE 216 at all). **Your code MUST NOT modify the original list of required courses in any way!**

We have provided you with a set of sample "transcripts" (lists of courses and grades); the starter code includes an abbreviated list of required courses for the CSE major that you should use as the first argument to your function.

Examples:

Function Call	**Return Value**
`checkCourses(required, "transcript1.txt")`	`["CSE214", "CSE303", "CSE316", "CSE320"]`
`checkCourses(required, "transcript2.txt")`	`["CSE114", "CSE214", "CSE215", "CSE220", "CSE312", "CSE316", "CSE416"]`
`checkCourses(required, "transcript3.txt")`	`True`