UNIVERSITY OF SASKATCHEWAN
Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephine: (306) 966-4886, Facimile: (306) 966-4884

Assignment 1

Python Programming

Date Due: 17 January 2020, 8pm **Total Marks: 66**

Version History

- **09/01/2020**: Corrected the simple examples for A1Q3, as highlighted in the description for that question. The data files provided for A1Q3 were also replaced.
- **08/01/2020**: released to students

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- **Assignments are being checked for plagiarism.** We are using state-of-the-art software to compare every pair of student submissions. Plagiarism can include: copying answers from a web page, or from a classmate, or from solutions published in previous semesters. Basically, if you cannot delete your whole assignment and do it again yourself (given adequate time), it's not your work, so don't try to claim credit for it. Your success in this course depends on depends on what you can do, not on what you can hand in.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M.
- Read the purpose of each question. Read the Evaluation section of each question.
- Make sure your name and student number appear at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- Do not submit folders, or zip files, even if you think it will help. It might help you, but it adds an extra step for the markers.
- Programs must be written in Python 3.
- **Assignments must be submitted to Moodle.** If you are not sure, talk to a Lab TA about how to do this.
- **Moodle will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded. **Do not send late assignment submissions to your instructors, lab TAs, or markers.**

Question 1 (25 points):

Purpose: To build a program and test it, starting with a design document. To get warmed up with Python, in case you are using it for the first time.

Degree of Difficulty: Moderate. Don't leave this to the last minute. The basic functionality of the various parts of this question are easy. There are aspects of some parts that you won't appreciate until you start testing.

References: You may wish to review the following chapters:

- General: CMPT 145 Readings, Chapter 1
- Functions Review: CMPT 141 Readings: Chapter 4
- Testing Review: CMPT 141 Readings: Chapter 15

Note: A1Q2 asks about your work on A1Q1

Keep track of how long you spend working on this question, including time spent reading the question, reviewing notes and the course readings, implementing the solution, testing, and preparing to submit. Also keep track of how many times you experienced unscheduled interruptions, delays, or distractions during your work; include things like social media, phone calls, texting, but not such things as breaking for a meal, exercise, commuting to or from the university. In A1Q2 you will be asked to summarize the time you spend working on A1Q1, and your answer will be more helpful to you if you have more precise records.

Background

A *Magic Square* is an arrangement of numbers in a square, so that every row, column, and diagonal add up to the same value. Below are two squares, but only one of them is a Magic Square.

8	1	6
3	5	7
4	9	2

1	9	6
5	3	7
4	8	2

The square on the left is a 3×3 magic square, whose rows, columns, and diagonals all sum to 15. On the right, is a 3×3 square of numbers whose rows columns and diagonals don't have the same sum.

There are magic squares of all sizes (except 2×2), but we'll be concerned with 3×3 squares, and checking if a given arrangement of 9 numbers is a magic square or not.

Definition: A 3×3 magic square is formally defined by the following three criteria:

- It contains the integers 1 through 9 inclusively.
- Every integer in the range 1 through 9 appears exactly once.
- Every row, column, and diagonal sums to 15.

In this question you will implement a program that does the following:

- It asks the user for a sequence of 9 numbers from the console. The order of the numbers is important, as the rows of the grid use this order. **For simplicity, assume that the user will type only integers on the console. For this question, you don't have worry about what to do if the user types anything other than integers.**

- It checks whether the sequence of integers is a magic square or not. Your program should display the message `YES` if it's magic, or `NO` if it's not.

It's very important to point out that **you are not being asked to construct a magic square; only to check if a square is magic or not.**

Task

We've given you a design document called `MagicSquareDD.txt` (available on Moodle), which is the end result of a fairly careful design process. It describes a collection of algorithms which, when used together, should solve the problem. **You must implement the program according to the design in this document.** There will be some very small decisions you still have to make about the implementation. Every "Algorithm" in the design document will be a procedure or function written in Python. Function names should be very strongly similar to the given design, if not identical. Many of the Algorithms have a small number of test cases which you should use to check your implementation; you do not need to create formal test cases for Algorithms that do not have test cases given.

It's a top-down design, but the algorithms are presented in the order they should be implemented. A good rule of thumb is this: "*Design top-down. Implement bottom-up. Test each function before implementing the next.*" It's not a hard and fast rule, but it's a good guide.

Here's how you should work with every Algorithm in the design document:

1. Read the design specification for the function.
2. Write a trivial Python function that does nothing, but has the correct name, the appropriate parameters, and a return value that's appropriate. For example:

```
def check_diagonals(square):
    '''
        square: a 3x3 list of integers
        Return: True if all the diagonals sum to False otherwise
    '''
    return False
```

   The function has the right parameters, and returns a value of the right kind, but doesn't really do what it is supposed to do, yet.

3. Implement the test cases for the function in a Python file named `a1q1_testing.py`. Yes, do that before you implement the function!
4. Run your test program. Some, if not all, of the tests should fail, because you haven't implemented the function yet. That's exactly where you want to be.
5. Implement the function carefully. Then run the test program. Debug this function as necessary until all the tests pass.
6. Go on to the next function assured that your implementation of the current function is completely correct. Do not disable the testing of any function. Every test for every function you've written should be tried every time you add a new function. That way you know if you changed something for the worse.

Because everyone is starting with the same design document, every program will have a high degree of similarity. That's perfectly okay. The value of this exercise is in the experience you gain from it, not the answer.

What to Hand In

- Your implementation of the program: `a1q1.py`.
- Your test program `a1q1_testing.py`

Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

- 5 marks: Your program `a1q1.py` follows the design document given.
- 5 marks: Your program `a1q1.py` is well-documented with doc-strings describing parameters, and return values.
- 5 marks: Your test program `a1q1_testing.py` includes all the test cases given in the design document.
- 5 marks: Your test program `a1q1_testing.py` runs without producing any error.
- 5 marks: Your program `a1q1.py` works as described above.

Question 2 (12 points):

Purpose: To reflect on the work of programming for Question 1. To practice objectively assessing the quality of the software you write. To practice visualizing improvements, without implementing improvements.

Degree of Difficulty: Easy.

Textbook: Chapter 3

Answer the following questions about your experience implementing the program in Question 1. You may use point form, and informal language. Just comment on your perceptions. Be brief. These are not deep questions; a couple of sentences or so ought to do it.

1. (2 marks) Comment on your program's *correctness* (see Chapter 3 of the textbook for the definition). How confident are you that your program (or the functions that you completed) is correct? What new information (in addition to your current level of testing) would raise your confidence? How likely is it that your program might be incorrect in a way you do not currently recognize?
2. (2 marks) Comment on your program's *efficiency* (see Chapter 3 of the textbook for the definition). How confident are you that your program is reasonably efficient? What facts or concepts did you use to estimate or quantify your program's efficiency?
3. (2 marks) Comment on your program's *adaptability* (see Chapter 3 of the textbook for the definition). For example, what if Assignment 2 asked you to write a program to check whether a 5×5 square was magic (bigger square with a larger sum, using the numbers 1 through 25)? How hard would it be to take your work in A1Q1, and revise it to handle squares of any size?
4. (2 marks) Comment on your program's *robustness* (see Chapter 3 of the textbook for the definition). Can you identify places where your program might behave badly, even though you've done your best to make it correct? You do not have to fix anything you mention here; it's just good to be aware.
5. (2 marks) How much time did you spend writing your program? Did it take longer or shorter than you expected? If anything surprised you about this task, explain why it surprised you.
6. (2 marks) Consider how often you were interrupted, distracted, delayed during your work for Question 1. Do you think these factors affected substantially increased the time you needed? If so, what kinds of steps can you take to prevent these factors?

What to Hand In

Your answers to the above questions in a text file called `a1q2.txt` (PDF, rtf, docx or doc are acceptable). Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

The purpose of these questions is to reflect on your experience. You are not expected to give the "right answer", or to have worked with perfection. Your answers are for you. We will give you credit for attempting to use this opportunity to reflect in a meaningful way, no matter what your answers are.

Each answer is worth 2 marks. Full marks will be given for any answer that demonstrates thoughtful reflection. Grammar and spelling won't be graded, but practice your professional-level writing skills anyway.

Question 3 (15 points):

Purpose: To build a program and test it, without starting with a design document. To get warmed up with Python, in case you are using it for the first time.

Degree of Difficulty: Moderate. Don't leave this to the last minute. This task involves an unfamiliar problem, but the program itself is not more difficult than anything you've done in CMPT 141.

References: You may wish to review the following chapters:

- File I/O Review: CMPT 141 Readings: Chapter 12

Note: A1Q4 asks about your work on A1Q3

Keep track of how long you spend working on this question, including time spent reading the question, reviewing notes and the course readings, implementing the solution, testing, and preparing to submit. Also keep track of how many times you experienced unscheduled interruptions, delays, or distractions during your work; include things like social media, phone calls, texting, but not such things as breaking for a meal, exercise, commuting to or from the university. In A1Q4 you will be asked to summarize the time you spend working on A1Q3, and your answer will be more helpful to you if you have more precise records.

The Torus Square

The following description considers the manipulation of a 3×3 array of integers, which can be considered a kind of simple game. The description will start by describing the rules of the game, and then you'll be told what your task will be. It is highly unlikely that you will discover anything sensible on the internet about this game. It was invented by one of your instructors.

Consider an 3×3 array of integers $\{0, \cdots, 8\}$ such as:

0	1	2
3	4	5
6	7	8

This is a kind of a puzzle, where we allow rows to be shifted left or right, and columns to be shifted up or down. We will call these actions *unit rotations*, for reasons that will be clarified immediately.

A *unit rotation* is simply a shift of all the elements of a row (or column) by one position. For example, we can rotate the top row of the previous example one position to the right to get the following:

2	0	1
3	4	5
6	7	8

Notice that the integer 2 that was on the end of the row moved to the front after the rotation to the right. Because the number 2 moved to the front of the row, we call it a rotation. Similarly, we can rotate the bottom row one position to the left to get the following:

2	0	1
3	4	5
7	8	6

Notice that the integer 6 that was on the front of the row moved to the end after the rotation to the left.

Rotations of a row can be one position right, or one position left. Rotations of a column can be one position up, or one position down. When a column is rotated up, the value on the top of the column is moved to the bottom of the column; all other elements are moved one place upwards. Because we only consider one position left, right, up, or down, we call it a unit rotation. When a row (or column) is rotated, only that row (or column) is rotated; all other rows (or columns) remain unchanged.

These unit rotations can be performed in sequences, one after another, which result in the numbers moving around the puzzle array.

We will represent each possible rotation with a code, which we will motivate by example, and then define carefully after the example. A unit rotation of the first row to the right will be represented by the string `'R 0'`. The first letter `'R'` indicates that a row is rotated to the right. The digit `'0'` indicates that the first row is rotated.

In general, a rotation will consist of a single letter followed by a single digit.

- `'R i'` means rotate row i to the right one position.
- `'L i'` means rotate row i to the left one position.
- `'U i'` means rotate column i up one position.
- `'D i'` means rotate column i down one position.

Here, i has to be a valid row or column index, that is: 0, 1, 2.

Now that we have a code for the rotations, we can write sequences of rotations with this code.

Suppose you are given an array A_1 and and a sequence of unit rotations, for example (Note: this example was corrected 09/01/2020):

```
0 1 2
3 4 5
6 7 8
2
L 0
U 2
```

The first three lines tell us the starting position. The number 2 tells us that there are two rotations to perform on the starting position. There are two rotations given (row 0 to the right, followed by column 2 up), which are the rotations used in the preceding description.

If we perform the given unit rotations, the resulting position would be the following:

1	2	5
3	4	8
6	7	0

Task

You'll be given several data files consisting of one example per file. Each file will describe an initial configuration of the puzzle, and then a sequence of unit rotations. In general, an example file will look like this:

```
a0 a1 a2
a3 a4 a5
a6 a7 a8
N
```

```
r1
...
rN
```

The first three lines are the starting position of the puzzle array. Assume these will be numbers, in the range 0-8. The 4th line tells you the number of rotations to perform, and the remaining N lines describe each rotation (using the code above).

Your job is to read the file, perform the rotations, and display the array showing the effect of all the rotations done in the sequence given. For example, if the input file contains the following example (Note: this example was corrected 09/01/2020):

```
0 1 2
3 4 5
6 7 8
2
L 0
U 2
```

your program should read the file, perform the rotations, and finally display the result:

```
1 2 5
3 4 8
6 7 0
```

Data files On Moodle you will find 2 ZIP files for your use in this question:

- `a1q3Examples.zip` Contains 15 example files, with rotation sequences of different lengths. The filenames are `torus_N_i.txt`, where N tells you how long the rotation sequence is, and i is an index. There are 3 examples for each N.
- `a1q3Solutions.zip` Contains 10 files that show the solutions for most of the examples. The solution file `torus_N_i_solution.txt` matches the example file `torus_N_i.txt`. We're not giving you solution files for all the examples, so that you have to be more thoughtful about assessing your work.
- Note: On 09/01/2020 these files were replaced, and new files were provided. If you downloaded them before this date, discard those and download the new ones.

You should be able to open ZIP files using software on any lab computer, and on your personal computer as well. If not, the ZIP format is standard, and File Compression utilities are easily downloaded from your favourite app store.

What to Hand In

- Your implementation of the program: `a1q3.py`.
- If you wrote a test script, hand that in too, calling it `a1q3_testing.py`

Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

- 10 marks: Your program works.
- 5 marks: Your program is well-documented.

Question 4 (14 points):

Purpose: To reflect on the work of programming for Question 3. To practice objectively assessing the quality of the software you write. To practice visualizing improvements, without implementing improvements.

Degree of Difficulty: Easy.

Textbook: Chapter 3

Answer the following questions about your experience implementing the program in Question 3. You may use point form, and informal language. Just comment on your perceptions. Be brief. These are not deep questions; a couple of sentences or so ought to do it.

1. (2 marks) Comment on your program's *correctness* (see Chapter 3 of the textbook for the definition). How confident are you that your program (or the functions that you completed) is correct? What new information (in addition to your current level of testing) would raise your confidence? How likely is it that your program might be incorrect in a way you do not currently recognize?
2. (2 marks) Comment on your program's *efficiency* (see Chapter 3 of the textbook for the definition). How confident are you that your program is reasonably efficient? What facts or concepts did you use to estimate or quantify your program's efficiency?
3. (2 marks) Comment on your program's *adaptability* (see Chapter 3 of the textbook for the definition). For example, what if Assignment 2 asked you to write a program to work with a Torus rectangle of 5×7)? How hard would it be to take your work in A1Q3, and revise it to handle arrays of any size?
4. (2 marks) Comment on your program's *robustness* (see Chapter 3 of the textbook for the definition). Can you identify places where your program might behave badly, even though you've done your best to make it correct? You do not have to fix anything you mention here; it's just good to be aware.
5. (2 marks) How much time did you spend writing your program? Did it take longer or shorter than you expected? If anything surprised you about this task, explain why it surprised you.
6. (2 marks) Did you create a design document, along the lines of the one provided for A1Q1? If so, how long did that take? If you did, do you think the time you spent resulted in a shorter total time for this question? If you did not create a design document, do you think I could have helped you?
7. (2 marks) Consider how often you were interrupted, distracted, delayed during your work for Question 3. Do you think these factors affected substantially increased the time you needed? If so, what kinds of steps can you take to prevent these factors?

What to Hand In

Your answers to the above questions in a text file called `a1q2.txt` (PDF, rtf, docx or doc are acceptable). Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

The purpose of these questions is to reflect on your experience. You are not expected to give the "right answer", or to have worked with perfection. Your answers are for you. We will give you credit for attempting to use this opportunity to reflect in a meaningful way, no matter what your answers are.

Each answer is worth 2 marks. Full marks will be given for any answer that demonstrates thoughtful reflection. Grammar and spelling won't be graded, but practice your professional-level writing skills anyway.