



Assignment 1 – Solutions and Grading

Basic Python, Chapters 1 and 2

Date Due: 19 May 2021, 11:59pm

Total Marks: ??

Question 1 (22 points):

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

Degree of Difficulty: **Easy.** This is just for fun, and to emphasize some of the ideas of Chapter 1.

References: You may wish to review the following chapters:

- General: CMPT 145 Readings, Chapter 1
- File IO Review: CMPT 141 Readings: Chapter 12

Restrictions: This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

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. Maybe go and read Q2 before you start Q1?

A simple calculating language: Tho The end goal of this question is a Python program that *interprets* a very simple calculating language called Tho. Don't worry, this is easy!

Your Python program will be able to open a file, read the commands that are in the file, perform the calculations, and will output the numerical result. The file can be considered a program or a script in the Tho language, and your Python program that reads the Tho script will be considered a very simple Tho interpreter.

The Tho language is very limited, and that's what makes it viable for a first question on a first assignment.

Tho Register The language is not like Python. In Python, we can use as many variables as we need. The Tho language will use exactly one register, called *R*, whose value can be modified by the Tho commands. At the beginning of every script, this register is initialized to be zero.

Tho commands The commands of the Tho language are not like in Python, which has assignment statements and expressions, and loops and functions, etc. In Tho, we only have very simple commands. Each command is rather like an instruction in assembly language, for a very simple kind of CPU.

To describe the commands, we have to give the syntax of the commands, and then clearly describe the behaviour or effect of the commands. In the description that follows, the command keywords will be written in upper case, e.g., **ASK**. Commands sometimes take an argument, which depends on that the script is trying to do. For arguments, we will use italics, e.g., *T*.

There are only two kinds of arguments for a Tho script. Usually, an argument will be a floating point number. Sometimes, a Tho command will use a name as an argument. A name is simply a letter or word, similar to names in Python.

We'll start with the **ASK** and **TELL** commands, which Tho uses to interact with the user.

Command	Effect
ASK <i>T</i>	Acquire a value from the console, and store it as the value of the name <i>T</i>
TELL	Display the value of <i>R</i> on the console

The effect of the **ASK** *T* command requires a bit more detail:

1. Display a simple prompt to the user, who will then type in a floating point value, followed by a return.
2. The user's value will be stored as the value for *T*.

We can say that the **ASK** command *creates* the name, and records its value. After the **ASK** command, the name can be used in other commands. It's important to understand that a name in Tho is not a variable; the value associated with a name created by the **ASK** command cannot be changed. It's a named constant that can be used elsewhere in the Tho script.

The **TELL** command displays the value stored in the register to the console. The syntax of this command is simply the word **TELL** on a line by itself.

The Tho language will also have 4 arithmetic commands:

Command	Effect
ADD <i>e</i>	$R = R + e$
SUB <i>e</i>	$R = R - e$
MUL <i>e</i>	$R = R \times e$
DIV <i>e</i>	$R = R / e$ (floating point division)

In this table, the *e* represents a simple expression. In the Tho language, a simple expression is either a floating point literal, like 3.1415, or a name created by an **ASK** command. For example, if the script has the command **ADD** 4, then the value 4 is added to whatever number is already in the register *R*. Notice that the commands mention the expression *e*, but not *R*. That's because all the arithmetic commands affect *R*, so we don't need to specify that in the command itself; the *R* is implied.

Tho scripts Now that we have seen the components of the Tho language, let's look at a script in this language. Here's an example:

```
ASK A
ASK B

ADD A
ADD B

DIV 2.0

TELL
```



The first two lines ask the user for two floating point values (using console input). Then the two values are added to the register, and the value in the register is divided by 2. Finally, the script displays the contents of the register on the console. Suppose the user entered the values 4 (for A) and 7 (for B).

Here's a transcript of interpreting the above Tho script:

```
File to load: sqrt.txt
A? 4
B? 7
5.5

Process finished with exit code 0
```

Notice that the ASK command prompts the user for input using the name from the ASK command.

The assignment page on Moodle will have a bunch of example scripts. Part of your work is to understand the Tho language so well that you can read the scripts and understand what they do.

Task

Write a program that:

- Asks the user for the name of a script to interpret.
- Opens the named script (it's a text file with commands in it), reading the contents.
- Executes the script line by line.

Hints:

- A Python file has been given to you containing the function `isfloat()`, which you can import. It takes a string as input, and returns `True` if the string can be converted to a floating point value, and `False` if not. The function itself uses exceptions, which you don't need to understand today; it's a topic we cover in second year.
- You will need to read a file using File IO. You will have to look at each line, to see what the command is, and depending on the command, you may need to look at an argument.
- For the ASK command, use a dictionary to remember the names and values. The keys in your dictionary will be the names, and the values are the floating point values that the user typed in. The ASK command affects this dictionary, but not the register.
- The arithmetic commands modify the value in the register *R*.
- Your main task is to correctly interpret correct Tho scripts. However, just as it is possible to write Python scripts that have syntax errors or logic errors, it is quite possible to write Tho scripts with syntax errors and logic errors. For this assignment, you are not required to deal with errors in the Tho scripts. If your program crashes because of an error in a Tho script (e.g., `DIV 0.0`), that's perfectly acceptable for this question. If your program crashes because of an error in your Python program, that's something you have to fix!
- The name of this language: `'PYTHON' [2:5]`

What to Hand In

- Your implementation of the Tho interpreter: `a1q1.py`.

Do not submit the example Tho scripts.

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

Evaluation

- (2 marks) Your Python program opens a Tho script file correctly, and reads its contents.
- (12 marks) Your Python program implements the commands:
 - ASK T : correctly interacts with the user, and stores the value
 - TELL: correctly displays the value in the register R
 - ADD e : correctly adds e to the value in register R
 - SUB e : correctly subtracts e from the value in register R
 - MUL e : correctly multiplies the value in register R by e
 - DIV e : correctly divides the value in register R by e
- (4 marks) Your program is well-documented according to the standards of CMPT 141.
- (4 marks) Your program correctly interprets all the test scripts used by the markers (which you do not have).



Solution: The Solutions folder contains the implementation of this application in the file `a1q1.py`. But let's go over some of the key features. We start by importing the provided module:

```
import isfloat as isfl
```

A module can be any Python script at all. To keep things simple, we are importing a script in the same folder.

Next, we open and read a file. Note that the filename is provided by the user:

```
filename = input("File to load: ")  
file = open(filename)
```

If this is not familiar to you, consult the CMPT 141 Readings, Chapter 12.

Now, we read the whole file all at once, storing it in a list. The following code does this in two stages. First we read all the lines, no matter what they are. Second we split each line into a list of strings.

```
script = [line.rstrip() for line in file]  
script = [line.split() for line in script if line]
```

If `rstrip()` or `split()` are not familiar to you, consult the CMPT 141 Readings, Chapter 12. These are very common tools, which you will use repeatedly. The above lines make use of list comprehensions, which were covered in CMPT 141. These are very useful, and you can't really say that you have learned Python without them.

The file should be closed, before we proceed. This is standard procedure, and it's easy to forget. In a professional application, files that do not get closed on purpose will get closed by accident, and that accident could result in data loss. So practice closing your files!

```
file.close()
```

Before we go further, we prepare for the fun by initialization of the variables we will use for the register and the named constants:

```
register = 0  
constants = {}
```

Note that variable `constants` is the name for an empty dictionary. Python dictionaries are tremendously valuable tools, which were covered in CMPT 141.

Now the fun begins. Our task is to look at each line, and do what the line says. We'll use a for-loop:

```
for c in script:
```

Because of the way the data in the file was processed (above), each line is now a list containing 1 or 2 strings. Our task is to look at the list, and decide what to do. We'll start by looking at the current command `c`, to see if it's an ASK command:

```
if c[0] == "ASK":  
    # ask the user for the value to use  
    temp = float(input(c[1]+"? "))  
  
    # record the name and value  
    constants[c[1]] = temp
```



Notice that if it is an ask command, then we use Python's `input()` to get a data value from the user. The prompt here is pretty terse. It's okay to be less terse. Notice also that the name and the value are stored in the dictionary for later use.

The TELL command is pretty simple, just send the value of the register to the console:

```
elif c[0] == "TELL":  
    print(register)
```

Now, for the arithmetic operations, they are all very similar. There are many ways to manage these similarities, and in the code below, I have tried to separate everything in common from everything that's different. The common code checks to see if we're given a literal value, or a name, after the arithmetic label:

```
if isfl.isfloat(c[1]):  
    argument = float(c[1])  
else:  
    argument = constants[c[1]]
```

After that, we just need to do the right thing to the register, i.e., add the argument for an ADD command, etc.

We've constructed a very simple interpreter for a very simple language. We could imagine adding more features to this language, making it more useful, but this is where we will leave it.



Notes for markers: The rubric usually breaks the marking into three parts:

- Give full marks unless you see a problem.
- Give partial credit if there is a minor problem.
- Give zero marks only if there is a serious problem.

Using this general guideline, assess the rubric on each point below:

- (2 marks) Your Python program opens a Tho script file correctly, and reads its contents.
- (12 marks) Your Python program implements the commands:
 - Deduct marks if you can see errors in the code.
 - * ASK T : correctly interacts with the user, and stores the value
 - * TELL: correctly displays the value in the register R
 - * ADD e : correctly adds e to the value in register R
 - * SUB e : correctly subtracts e from the value in register R
 - * MUL e : correctly multiplies the value in register R by e
 - * DIV e : correctly divides the value in register R by e
- (4 marks) Your program is well-documented according to the standards of CMPT 141.
 - Give zero marks if there is no documentation.
 - Give full marks if there is some documentation, but not too much.
 - Give partial credit if there is too much documentation, or too little.
- (4 marks) Your program correctly interprets all the test scripts used by the markers (which you do not have).
 - Drop the given script beside the student's work (a folder is created for each student when you down load the submission from Moodle).
 - Run Python from a command-line, and give the name pf the script when prompted.
 - If the program does not run, or does not ask for the file, give zero marks.
 - If the program asks for a file name, and the answer is correct, give full marks.
 - Give partial credit if you can think of a reason not to give full marks or zero.

Question 2 (6 points):

Purpose: To reflect on the work of programming for Question 1.

Degree of Difficulty: Easy.

Textbook: Chapter 3

Restrictions: This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

Answer the following questions about your experience implementing the program in Question 1. Be brief. These are not deep questions. They are intended to get you to reflect on your experience, so you can learn from it.

1. (2 marks) How confident are you that your program is correct? What is the basis for your confidence? How likely do you think it is that your program might be incorrect in a way you do not currently recognize?
2. (2 marks) How much time did you spend writing your program? Did it take longer or shorter than you expected? Which part of this task took the longest?
3. (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.



Solution: There are no wrong answers, but marks will be deducted if your answers are not relevant to the issues at hand. Here are some comments, and the level of discussion we would expect.

1. I am confident that the program is correct, because it correctly runs the example scripts given in the assignment. There is an error that occurs from time to time when very large numbers are entered by the user (this error happens to be due to floating point arithmetic, and is not a programming error). There might be errors that occur when given a script that does something unusual, but it's hard to imagine what kind of a script that would be.
2. The implementation took about 15 minutes. This is not the first Python program I have written. The hardest part of this work was working out how to allow the arithmetic commands to use a literal number or a named constant.
3. During the time I wrote up these solutions, I was interrupted a few times by students. I don't use social media while I am in my office, with the exception of email, which is business. I think the interruptions caused a delay, but about what should be expected for a prof. Some profs close their doors, and do not allow students to interrupt them. I sometimes work early in the mornings when students are less likely to drop in. Today, I am writing these solutions at the end of a long day. Interruptions distract me more when I am tired than when I am focussed.

Notes for markers:

- This question is for students to practice thinking about these issues. There is no need for the answers to be positive. If the student knows their code is not very good, and says why, that's worth full marks here.
- Give zero marks for any part if no response was given, or if the response was completely irrelevant to the question.
- Give partial credit if the response seemed too superficial. A superficial response would be something like: "I'm sure my program is correct" without referring to test cases.

Question 3 (20 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:

- Lists: CMPT 145 Readings: Chapter 2.
- File I/O Review: CMPT 141 Readings: Chapter 12

Restrictions: This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

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 $\{0, \dots, 8\}$ 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 a sequence of unit rotations, for example:

```
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 prompt the user for a filename, read the named 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:

```
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 file-names 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.

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`.

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

Evaluation

- (2 marks) Your program correctly opens and reads the contents of a file containing an array, and a sequence of rotations.



- (8 marks) Your program correctly implements the 4 types of rotations.
- (2 marks) Your program correctly displays the resulting array on the console.
- (4 marks) Your program is well-documented according to the standards of CMPT 141.
- (4 marks) Your program produces correct outputs on all the test scripts used by the markers (which you do not have).



Solution: A solution is provided in the Solutions folder, along with a test script. Students should study the solution to see some clever ideas that they might use in other contexts.

Viable solutions can use a nested list, or a numpy array. A flat list is possible too, though I think it's rather difficult. Rotating rows and columns amounts to copying values from one part of the torus square to another, in an order that preserves the data.

A good implementation will have the following properties:

- Good use of functions that can break up the task into testable chunks. For example, the model solution had the following functions:
 - `rotate_row()`: Given a torus square, a direction, and a row index, mutates the torus square so that the rotation is performed.
 - `rotate_column()`: Given a torus square, a direction, and a column index, mutates the torus square so that the rotation is performed.
 - `read_file()`: Opens a named file, reads the initial square and the commands.
 - `run_commands()`: Given an initial torus square, and a list of commands, calls `rotate_row()` or `rotate_column()` appropriately.

Good implementations will generalize the copying to keep methods relatively short. A weaker implementation would result in having a special case for each possible command. Long programs are less preferable because they are harder to debug and to maintain in the future. Shorter programs are harder to design.

The model solution used functions that modified (mutated) the torus squares, and did not rely on return values. I did this deliberately to demonstrate the technique, but it's not a required technique. I think I prefer functions that return values for most things. But to understand how the model solution works, you need to understand references! So a good exercise for Chapter 3!



Notes for markers:

- This exercise is intended to allow students to do whatever they feel like doing to solve it. The only learning objective was to get a job done.
- Students were given freedom to implement the program in any way they wanted. There is no requirement for use of functions, or list-of-lists, or anything.
- Run the code only if you need to. Hopefully
- To assess the implementation: The rubric usually breaks the marking into three parts:
 - Give full marks unless you see a problem.
 - Give partial credit if there is a minor problem.
 - Give zero marks only if there is a serious problem.
 - (2 marks) Your program correctly opens and reads the contents of a file containing an array, and a sequence of rotations.
 - (8 marks) Your program correctly implements the 4 types of rotations.
 - (2 marks) Your program correctly displays the resulting array on the console.
 - (4 marks) Your program is well-documented according to the standards of CMPT 141.
 - * Give zero marks if there is no documentation.
 - * Give full marks if there is some documentation, but not too much.
 - * Give partial credit if there is too much documentation, or too little.
 - (4 marks) Your program produces correct outputs on all the test scripts used by the markers (which you do not have).
 - * Unfortunately, the instructions did not require that the program allow a choice of which example to try.
 - * Give full marks if the code looks good, or if you can't find anything wrong.
 - * Give partial credit if you can find minor errors by reading through the code.
 - * Give zero only if there's nothing handed in.

Question 4 (6 points):

Purpose: To reflect on the work of programming for Question 3.

Degree of Difficulty: Easy.

Textbook: Chapter 3

Restrictions: This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

Answer the following questions about your experience implementing the program in Question 3. Be brief. These are not deep questions. They are intended to get you to reflect on your experience, so you can learn from it.

1. (2 marks) How confident are you that your program is correct? What is the basis for your confidence? How likely do you think it is that your program might be incorrect in a way you do not currently recognize?
2. (2 marks) How much time did you spend writing your program? Did it take longer or shorter than you expected? Which part of this task took the longest?
3. (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.



Solution: Here are some responses to the questions, demonstrating the level of discussion we would expect. The comments are reflecting on the example solution provided for Question 3, which was probably correct, but poorly documented.

1. I tested the two rotation functions in all combinations, and I tested the full functionality on all the examples that had solutions given. See the test script! I am very confident that the program works, and I doubt more testing will find a bug that I have not found.
2. I think it took me about 30 minutes to complete the whole thing, including a pretty deliberate design process (iterative refinement, as in CMPT 141), and careful testing of my functions. I think that's about what I should have expected. I am too optimistic, though. I always under-estimate the time I need to do this work.
3. I was only distracted by a small number of emails from students and university folks. Not too bad. But the summer is pretty quiet. I was able to ignore the emails until I completed my task.

Notes for markers:

- This question is for students to practice thinking about these issues. There is no need for the answers to be positive. If the student knows their code is not very good, and says why, that's worth full marks here.
- Give zero marks for any part if no response was given, or if the response was completely irrelevant to the question.
- Give partial credit if the response seemed too superficial. A superficial response would be something like: "I'm sure my program is correct" without referring to test cases.