Assignment 1

Python Programming, and References

Date Due: May 23, 2019, 7pm **Total Marks: 40**

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.
- 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.
- 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.** There is a link on the course webpage that shows you 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.
- Read the purpose of each question. Read the Evaluation section of each question.

Version History

- **05/16/2019**: released to students

Question 1 (10 points):

Purpose: To work with the concept of references a bit more carefully.

Degree of Difficulty: Easy. If you understand references very well, this is not difficult.

In class (and in the readings) we saw a version of Selection sort. As described in the readings, our implementation of selection sort works by repeatedly removing the smallest value from an unsorted list and adding it to end of a sorted list until there are no more values left in the unsorted list (see Chapter 1 of the readings).

```
1  unsorted = [3, 2, 5, 7, 6, 8, 0, 1, 2, 8, 2]
2  sorted = list()
3
4  while len(unsorted) > 0:
5      out = min(unsorted)
6      unsorted.remove(out)
7      sorted.append(out)
8
9  print(sorted)
```

One problem with this implementation is that it modifies the original list (line 6). Unless we know for sure that a list will never be needed in the future, removing all contents is a bit drastic. To address this problem, we can change the code so that a copy of the original list is made first. Since we make the copy, we can say for sure that the copy will never be needed in the future, and so removing all its contents is absolutely fine.

The file `a1q1.py` is available on Moodle, and it contains a function called `selection_sort()` which is very similar to the above code:

```
1  def selection_sort(unsorted):
2      """
3      Returns a list with the same values as unsorted,
4      but reorganized to be in increasing order.
5      :param unsorted: a list of comparable data values
6      :return:  a sorted list of the data values
7      """
8
9      result = list()
10
11     # TODO use one of the copy() functions here
12
13     while len(acopy) > 0:
14         out = min(acopy)
15         acopy.remove(out)
16         result.append(out)
17
18     return result
```

On line 11, there is a TODO item, which is where we will add code to create a copy of the original unsorted list.

Also in the file are 5 different functions whose intended behaviour is to copy a list. Your job in this question is to determine which, if any, of these functions does the job right.

Note: There is a Python list method called `copy()`, which we are not using on purpose. We need to understand references, and we must not side-step the issue.

For each of the 5 `copy()` functions in the file `a1q1.py`, do the following:

- Determine if the function makes a copy of the list. Hint: some do not!
- Determine if the function is suitable for use in `selection_sort()`, that is, if it works, would you use this function? Do you think it is a good function? Hint: some are not!

Do not change the function `selection_sort()` except to make use of one of the versions of the `copy()` function on line 11. Do not change the code for any of the `copy()` functions, except to add a doc-string.

What to Hand In

Your answers to the above questions in the text file called `a1_reflections.txt` (PDF, rtf, docx or doc are acceptable). Please help the marker by clearly indicating the question number.

You might use the following format:

```
Question 1
copy1()
   - makes a copy
   - is suitable
copy2()
   - makes a copy
   - is not suitable
```

The above example does not necessarily reflect the right answers!

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

Evaluation

Each of the copy functions is worth 2 marks; 1 mark for each question.

Question 2 (10 points):

Purpose: To work with the concept of references a bit more carefully. To debug a program that uses references incorrectly.

Degree of Difficulty: Moderate. If you understand references very well, this is not difficult.

In the file named `histogram-broken.py`, you'll find code to display a primitive histogram on the console based on integers typed on the console by the user. Here is an example of how it might work:

```
How many data values? 10
Enter integer data

1 2 3 4 5 4 3 2 3 4

 ----------- Histogram ----------------

1: *
2: **
3: ***
4: ***
5: *
```

The Histogram is supposed to print one `*` for each occurrence of the value in the input. Histograms are useful for visualizing the distribution of data. For example, from this histogram, we can see that 3 and 4 were the most common values in the input (there were three of each).

Unfortunately, the implementation in the file we've provided has several errors, as if written by a novice programmer who didn't take the time to design the algorithm carefully, and so did not quite have enough time to finish.

Your task is as follows:

1. Study the functions carefully. While there are some errors, there are parts of the functions that are very close to working. The purpose of this question is not to get you to invent anything tricky, but to get you to practice debugging.
2. Figure out what each part of the functions is supposed to do. The programmer left no helpful comments to assist your study. Sorry.
3. Add doc-strings appropriate to the functions. You might have to modify the doc-strings if you change the functions.
4. Fix the obvious errors, and then try to get the implementation working by debugging and testing.
5. Make a list briefly mentioning every change you make to the program. For example you might have a list that starts like this:

```
1. Deleted line 7 of the original program.
2. Changed function isNegative() to return a Boolean value instead of a string.
```

Hints: We created these functions by starting functions that worked perfectly, and then we added errors to it, similar to the errors novices might make. Most of the errors are related to the material we've covered in lecture so far, including review and the unit on references. Careful study should be enough to fix this program. You won't have to add much code at all. Just fix what's there.

What to hand in:

- The list of changes you made to the program, in the text file called `a1_reflections.txt` (PDF, rtf, docx or doc are acceptable). This is the same document you used earlier. Please help the marker by clearly indicating the question number.
- Hand in your working program in a file called `a1q2.py`.
- A text-document called `a1q2_demo.txt` that shows your program working on a few examples. You may copy/paste to a text document from the console.
- If you wrote a test script, but this is not necessary, hand that in too, calling it `a1q2_testing.py`

Evaluation

- 4 marks. Your list of changes includes the bugs we know about in the original program.
- 3 marks. Your program works.
- 3 marks. Your functions have appropriate doc-strings, and other appropriate documentation.

Question 3 (10 points):

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

Degree of Difficulty: Moderate. There many ways to complete this problem, but don't leave this to the last minute. The basic functionality is easy. There are aspects of the problem that you won't appreciate until you start testing.

A mathematician named Blaise Pascal has been accused of murder, and Phoenix Wright, ace attorney, has taken the case as the defence attorney. A piece of paper with what appears to be a Pascal's Triangle written on it was found at the crime scene. If the paper does indeed have a correct Pascal Triangle on it (`EvidenceTriangle.txt` from moodle), then the defendent will be found guilty *(Prosecutor: 1. Only Pascal could successfully write such a large Pascal's triangle. 2. The triangle is named after Pascal, which is pretty incriminating if you ask me)*. However if the found Pascal's Triangle turns out to be incorrectly constructed, Pascal will be found innocent *(Phoenix: Pascal would* ***NEVER*** *make a mistake creating his triangle!)*. Phoenix isn't great at math, and needs your help for this case!

A Pascal's triangle is a triangular array of numbers. Typically, the top row is row 0, and for each row there are *(row number) + 1* numbers. Each number is the sum of the number above it and to the left, and the number above it and to right. The top row is special, and contains only a 1. See https://en.wikipedia.org/wiki/Pascal%27s_triangle for more information.

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
1   5  10  10   5   1
```

In this question you will implement a program that checks whether a series of numbers constitutes of a Pascal's Triangle or not. Your program should work by reading input from the console, and sending an answer to the console, as in the following example:

- It reads a number N on a line by itself, where N > 0. This will be the number of rows.
- It reads N lines of numbers, with each line needing *(row number) + 1* positive numbers (Remember row numbers start at 0).
- It checks whether the sequence of numbers is a Pascal Triangle or not. Your program should display the message "yes" if it satisfies the above criteria, or "no" if it does not.

4
1
1 1
1 2 1
1 3 3 1

5
1
1 1
1 2 1
1 4 3 1
1 4 6 4 1

The example above shows input entered by the user, with each newline being a new row in the table. The input from the left table should output yes whereas from the right should output a no, as on row 3 there's a 4 instead of a 3 (remember rows start at 0).

There are many ways to solve this problem. You are responsible for the design of your program.

Note: You may complete this question using any method you see fit, but you will have to reflect on your method in the next question.

Note: One option is to use an array of arrays, or list of lists. Before accessing a list or array, you may need to check that the index you are using is valid and within your bounds.

What to Hand In

- Your implementation of the program: `a1q3.py`.
- A text document called `a1q3_demo.txt`, showing at least six (6) demonstrations of your program working on 3 differently sized examples that are true Pascal Triangles, and 3 differently sized examples that are not Pascal Triangles. This is a demonstration, not testing. Finally, at the end of `a1q3_demo.txt` show the input and output of running the numbers from `EvidenceTriangle.txt` from moodle, and declare whether or not the defendant is GUILTY or NOT GUILTY! Note, depending on the text editor/line endings, you may be able to copy paste the input from `EvidenceTriangle.txt` into your command line.
- 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

- 5 marks: Your program works.
- 1 marks: Your program outputs the correct answer when fed in the input from `EvidenceTriangle.txt` (found on moodle).
- 4 marks: Your program is well-documented.

Question 4 (10 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.

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; you do not have to give really deep answers. 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. How confident are you that your program (or the functions that you completed) is correct? Would it work on extremely large Pascal Triangles, or would it fail due to hard-coding?
2. (2 marks) Comment on your program's efficiency. How confident are you that your program is reasonably efficient?
3. (2 marks) Comment on your program's reusability. For example, if another student from the class read your code, would s/he understand it in a reasonable time? Have you organized your code into functions?
4. (2 marks) Comment on your program's robustness. 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.
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.

You are not being asked to defend your program as being good in all these considerations. For example, if your program is not very robust, you can say that; you don't need to make it robust.

What to Hand In

Your answers to the above questions in the text file called `a1_reflections.txt` (PDF, rtf, docx or doc are acceptable). **This is the same document you used for Q1 and Q2.** Please help the marker by clearly indicating the question number.

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

Evaluation

Each answer is worth 2 marks. Full marks will be given for any answer that demonstrates thoughtful reflection. Grammar and spelling wont be graded, but practice your professional-level writing skills anyway.