Lab 6

CS161

For each of the problems below be sure to design your solution before you write any Python code. Follow the software development stages:

1. Analyze the problem
2. Determine the specifications for your solution
3. Create a design for your solution in pseudocode
4. Implement your design in Python
5. Test/debug your solution

Each Python module you turn in must begin with a block, called a *docstring*, that looks like the example below (see PEP 257 for more details). The module docstring must follow the shebang and coding statements in your module; i.e. it must be the first actual line of code in the module.

In your module docstring show the work you did following the software development stages above. Make notes about the problem as you analyze it; write specifications for what your program must do to find a solution; write your program out in pseudocode *before* you start to write Python; and design a set of test data that you can use to prove your program is producing correct results. Only after you've done all of this should you try to translate your pseudocode to Python.

```
#! /usr/bin/env python3
# coding=utf-8

"""
Brief, prescriptive, one sentence description of what the module does.

A more detailed explanation of what the module does. Use complete sentences and
proper grammar. Treat this as if it is documentation someone else will be reading
(because it is). Include any notes from your problem analysis as well as your
program specifications here.

Pseudo code, to be written before any actual Python code.

Assignment or Lab #

Firstname Lastname
Firstname Lastname of partner (if applicable)
"""
<two blank lines>
<your code>
```

Every function or method you write must also include its own docstring. That header will also use triple quotes and begin with a prescriptive, one sentence description of what the function or method does. If the function or method takes arguments, they should be described as well as any non-obvious return value(s). See examples below or PEP 257 for more details).

```
def print_greeting():
    """Print a greeting to the user."""
    print("Hello out there in userland!")


def hypotenuse(side_a, side_b):
    """
    Calculates the length of the hypotenuse of a right triangle using
    the formula c^2 = a^2 + b^2.

    side_a – the length of side A of the triangle
    side_b – the length of side B of the triangle
    """
    c = a**2 + b**2
    return math.sqrt(c)
```

Pair Programming

In this lab we are going to use **pair programming**.

> *Pair programming is an agile software development technique in which two programmers work together at one workstation. One, the **driver**, writes code while the other, the **observer** or **navigator**, reviews each line of code as it is typed in. The two programmers switch roles frequently.*
>
> *While reviewing, the observer also considers the "strategic" direction of the work, coming up with ideas for improvements and likely future problems to address. This is intended to free the driver to focus all of their attention on the "tactical" aspects of completing the current task, using the observer as a safety net and guide.*

Choose one partner in the lab to work with. You can pair program in one of two ways:

1. Two people can work at one computer, occasionally switching the driver and observer roles. It is *critical* that both members be engaged in the work. You should take turns being in the driver and observer roles with only the driver typing and the observer making comments or driving the direction of the code.
2. Two people can work at separate machines using Visual Studio Live Share. Live Share, available as a free plug-in for VS Code and already installed on the school's Surface devices, enables you and your partner to collaborate on the same codebase without the need any difficult configuration. When you share a collaborative session, your partner sees the context of the workspace in their editor. This means they can read the code you shared without having to clone a repo or install any dependencies your code relies on. They can use rich language features to navigate within the code; not only just opening other files as text but using semantic analysis-based navigation like Go to Definition or Peek. Again, you should take turns being in the driver and observer roles with only the driver typing and the observer making comments or driving the direction of the code.

You will have **two** weeks to complete this lab.

Exercise 1

As a reminder, you are to be using *pair programming* when working on this lab. Resist the temptation to split the work and have each group member complete an exercise on their own. Doing so is against the spirit of the assignment and would be considered cheating. I know this may mean having to meet outside of class, physically or virtually, but you will gain valuable insight working with someone else.

The goal of this part of the lab is to practice with Lists. In Python, the List type is a container that holds a number of other objects, in a given order. This data structure is so useful and powerful that you can find it in almost every algorithm.

In this lab, we want to generate a Pascal's triangle. You can get the detailed explanation from Wikipedia (wikipedia.org/wiki/Pascal's_triangle), and the figure below is a simple Pascal's triangle whose height is 7.

```
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
1 6 15 20 15 6 1
```

You are going to write a program that asks for the height of Pascal's triangle, then generate the triangle and output it in the same style as above example.

Pascal's triangle rules:

1. To generate the triangle, you start from the first row, which is always 1, and the second row which is always 1 1.
2. After the first two rows, each row at level *h* is generated from the values at row *h*-1. Note that the leftmost number and the rightmost number in any row are always 1. Note that, in row *h*, there are *h* numbers.

Part 1

(A) Write a function named `make_new_row` that takes one argument `old_row` and generates the next row of Pascal's triangle (starting with `old_row = [1,1]`). On the class web page is linked a file named `lab_6_stub.py` that contains a function header and requirements. For example:

```
>>> make_new_row([1,1])
[1, 2, 1]
>>> make_new_row([1,2,1])
[1, 3, 3, 1]
>>> make_new_row([1,3,3,1])
[1, 4, 6, 4, 1]
>>> make_new_row([1,4,6,4,1])
[1, 5, 10, 10, 5, 1]
```

Hints:

1. Append new values to a `new_row` list. After the initial 1, the values will be the sum of two elements of `old_row`.
2. It is easiest to use a `for` loop that loops through indices, e.g. for `i in range(n):` so you can refer to adjacent items using indices `i` and `i + 1`, but be careful that you do not get too large an index, i.e. out of range.
3. Don't forget the last 1.

(B) Refactor your function to handle the following special cases:

1. If `old_row` is `[]`, then return `[1]`.
2. If `old_row` is `[1]`, then return `[1,1]`.

Adjust the contract comments to reflect handling of special cases.

Part 2

Write a program that prompts for a height greater than 0 and prints lists of Pascal's triangle rows to that height (no error checking needed). For example:

```
>>>
Enter the desired height of Pascal's triangle: 7
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
```

Part 3

Rewrite your program so that you collect all the rows into a master list, that is, a list of lists. You can do that by starting with a list and then appending lists onto the list. Print your list of lists two ways:

1. If your master list is named `triangle`, simply `print(triangle)`
2. Then print each list on its own line

The output is shown below:

```
Enter the desired height of Pascal's triangle: 7
Printing whole list of lists:
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10,
10, 5, 1], [1, 6, 15, 20, 15, 6, 1]]
Printing list of lists, one list at a time:
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
```

Finally, print Pascal's triangle as shown at the beginning of this exercise; i.e., with each list centered and without either commas or square brackets. A good way to do this is to convert each list to a string and then prints the strings.

Write and save your code in a file named `pascal.py` for full credit.

Exercise 2

This assignment will give you experience working with lists, file I/O, and exceptions as well as more experience with functions.

Background

The Centers for Disease Control (CDC) keeps data on what it calls "Winnable Battle Risk Factors and Health Indicators." These are negative behaviors or incidents that could possibly be avoided by changes in lifestyle. In this project you will examine some CDC data to discover the states with the best and worst records in regards to a few of these risk factors and health indicators.

Project Specifications

The file `riskfactors.csv` lists data on 20 different risk factors and health indicators for each state. The data is in "comma separated value" format (csv), which means that each entry is separated from the others by a comma. Examining the file in a text editor or a spreadsheet program or both should help you understand the format. Although not necessary to complete this lab, the Real Python article Reading and Writing CSV Files in Python may prove helpful.

Since analyzing 20 different indicators can be a bit confusing, we will only look at five: Heart Disease Death Rate, Motor Vehicle Death Rate, Teen Birth Rate, Adult Smoking, and Adult Obesity. Your program will read in the data from the csv file and find the states with the best and worst record for each of these indicators (largest and smallest values). It will produce a file called `best_and_worst.txt` which lists the states that have the highest and lowest value for each of the indicators, along with their values.

Your program must be general enough to work with a similar file with states as rows and with the same column headers. That is, if the CDC puts out a new file with different values in the cells (e.g. maybe new research changed some values), your program will work correctly.

Additional Requirements

1. You must create and use at least 2 meaningful functions (your choice).
2. Prompt for the input file. Your program must check if the file exists. If it does not, your program should output a "file not found" message and keep asking until a correct file is entered. (You do not need to prompt for the output file name.)
3. Allow the user to provide the name of the files to read and write as arguments at the command prompt using either `argparse`, `getopt`, or `click` modules. If they provide only one file, assume it is the one to read and prompt for the name of the file to write to. If they provide neither, prompt for both.

4. Your program must format the file into columns (see below). You don't have to match the formatting exactly, but columns should line up and it should be readable. Use string formatting (this StackOverflow question may be helpful).

Program output:

```
Indicator                      : Min                          Max
-------------------------------------------------------------------------------
Heart Disease Death Rate (2007)  : Minnesota           129.8    Mississippi    266.5
Motor Vehicle Death Rate (2009)  : District of Columbia  4.8    Wyoming         24.6
Teen Birth Rate (2009)           : New Hampshire        16.4    Mississippi     64.2
Adult Smoking (2010)             : Utah                  9.1    West Virginia   26.8
Adult Obesity (2010)             : Colorado             21.4    Mississippi     34.5
```

Assignment Notes

1. Not all lines of the file contain state data. In particular, the first few lines are irrelevant.
2. Don't forget to convert strings to numbers where appropriate.
3. Watch out for the fact that some data has a percent sign.
4. To open a file for output, remember:
    a. Open the file with the `'w'` mode string.
    b. You can only write strings to a file, so you must convert each output to a string before you write them.
    c. Also, remember that if you want a separate line to occur in your output file, you must specifically output the carriage return/line feed string `'\n'`.
5. Don't forget to close your file—otherwise the string might not get written. If you find that your file has nothing in it or is missing information, forgetting to close the file is a likely reason.
6. Depending on how you design your program you may find some, but not all, of the following list functions and methods useful: `sort`, `sorted`, `min`, and `max`.
7. There exists a module for reading csv files. This csv file is formatted nicely so that using the csv module is not needed—it is an unnecessary complication. However, you are free to use it, if you wish—it is a wonderfully powerful module that is necessary for dealing with most csv files.

Write and save your code in a file named `cdc.py` for full credit.

Submitting Your Lab

Submit all your Python files, and any other necessary files, in the appropriate folder in Google Drive as demonstrated by your instructor.