Lab 06: File I/O

Goals for this lab:

1. Read in data from a file
2. Write data to a file
3. Define and call functions
4. Pass data to a function
5. Return data from a function
6. Divide a program into multiple files
7. Write and execute simple Python programs using the given instructions

General Guidelines:

- Use meaningful variable names
- Use appropriate indentation
- Use comments, especially for the header, variables, and blocks of code.

Example Header:

```
# Author: Your name
# Date: Today's date
# Description: A small description in your own words that describes what
the program does. Any additional information required for the program to
execute should also be provided.
```

Example Function Docstring:

```
"""
General function description
:param p1: DESCRIPTION
:type p1: TYPE
:param p2: DESCRIPTION
:type p2: TYPE
:return: DESCRIPTION.
"""
```

1. Averages

Input data read from the keyboard and output data written to the screen are temporary and will be lost when the program ends. Files then provide a long-term data storage solution. The reading from/writing to a file in a program is usually referred to as file I/O (Input/Output). However, each language handles file I/O a little differently in terms of how files are accessed, what functions are available, and how data must be formatted coming into the program and going out to the file.

Before we get started writing the program, using a test editor of your choice (Notepad, TextEdit, etc) create a text file called **numbers.txt** that will contain the following integer values, each on a separate line as follows:

```
13
42
19
352
17
652
53
84
35
```

Working with files consists of the following three steps:

a. Open the file.
b. Process the file (includes reading from and/or writing to the file).
c. Close the file.

You will then write a complete Python program called **averages.py** that will read from this file, compute the average of these integers in the file (reading from the file), and then print out the average of these integers. Make sure you create a `main()` function and include all of the functionality for this lab inside of it, and don't forget to call the function at the end of the program! In order to read, process, and close the file, first need to open the file for reading (no need to prompt the user for the file name this time) and create any necessary variables so you can computer the average later. When processing the file, you can use either a `for` loop or `while` loop to read in one line at a time, but remember that both approaches need to be setup slightly differently. Once you read in a line, be sure to use the `strip()` function to remove any newlines from the end of the line. Next, print the line to the screen, and then parse the value and store it as part of your running total. After you have read in all of the lines in the file, be sure to close the file, compute the average, and then output that average to the screen. That's it!

Make sure you save your **averages.py** as you will be submitting this file for Lab 06.

2. Poetic

For this part of the lab, you will first need to create an input file named **Snowball.txt** that stores the following text:

Snowball
Shel Silverstein
I made myself a snowball
As perfect as could be.
I thought I'd keep it as a pet
And let it sleep with me.
I made it some pajamas
And a pillow for its head.
Then last night it ran away,
But first-it wet the bed.

In this file, the first line is the title of the poem, the second line is the author, and all other lines are the poem itself.
To complete the program:

- In a file named **poem.py** , you will need to read in a poem, store the information, and then write a short summary of the poem to another file. Your program should contain:
    - Your `main` function, which should:
        - Contain a variable to store the poem's title
        - Contain a variable to store the poem's author
        - Contain a `List` variable to store the lines of the poem
        - Contain a variable to store the name of the file the poem is in
        - Prompt the user for the name of the file they wish to have read in and summarized and store it in a variable
        - Test if the file exists using the appropriate function from the `os` module, and if the specified file does not exist, the program should repeatedly reprompt the user for a new file name until they provide a correct one
        - Open the file for reading
        - Read and store the name of the poem
        - Read and store the author of the poem
        - Read each line of the poem and store it in the `List`
        - Close the file
        - Open a file named **Output.txt** for writing
            - If the file does not already exist, `open()` will create it
        - Using appropriate messages output the following information to **Output.txt**
            - The name of the poem
            - The author of the poem
            - The number of lines in the poem
            - The first three lines of the poem to provide a preview of the poem
        - Close the file

Make sure you save **poem.py** as you will be submitting this file for Lab 06.

3. Print By Numbers

For this part of the lab, you will be reading in a file containing numbers, separated (delimited) by commas and converting those numbers into specific symbols that will create an ASCII art image. This will require the creation of multiple functions that feed information into each other. Additionally, you will need to separate the program into two files, **paintByNumbers.py** and **pbnFunctions.py** , such that **paintByNumbers.py** contains your `main` function and **pbnFunction.py** contains all of your other functions. Also, be sure to download the **image.csv** file to use for this part of the lab.

To complete the program create a file named **pbnFunctions.py** and inside:

- Define a function named `getFileName()` that has no parameters
    - Prompt the user for and store the name of a file
    - Test if the file exists using the appropriate function from the `os` module, and if the specified file does not exist, the program should repeatedly reprompt the user for a new file name until they provide a correct one
        - Do not forget to import the `os` module!
    - Return the name of the file
- Define a function named `convertLine()` that has a parameter named `line`, that represents a line in the file that needs to be converted from numbers into symbols
    - Use the `strip()` function to remove any newlines from the end of `line`
    - Create a new variable called `newLine` to store a String that will be build by converting numbers into symbols
    - Use the `split()` function to convert `line` into a `List` of numbers, and specify the delimiter to be a comma
        - Make sure you store the result of the `split()` function!
    - Iterate through all of the values in the `List` of numbers and using `if`/`elif` statements converting the following numbers into the following symbols:
        - 1 becomes a space
        - 2 becomes a ,
        - 3 becomes a _
        - 4 becomes a (
        - 5 becomes a O
        - 6 becomes a)
        - 7 becomes a -
        - 8 becomes a “
    - Return `newLine`
- Define a function named `processFile()` that has a parameter named `filename`, that represents the name of the file to be read in
    - Using `filename`, open the input file for reading
    - Open the output file `painting.txt` for writing
    - Using a loop of your choice, read in each line in the input file and store it in a variable named `line`
    - Call your `convertLine()` function and pass it `line`, storing the result in a variable named `newLine`
    - Write the contents of `newLine` to the output file on a new line
    - Once all of the lines in the input file have been read in, processed, and written to the output file, close both files

Create a file named **paintByNumbers.py** and inside:

- Define a function named `main()` that has no parameters
  - Call your `getFileName()` function and store the returned filename in a variable
  - Call your `processFile()` function and pass it the variable
  - Inform the user that their image is located in the file painting.txt
- Do not forget to import the `pbnFunctions` module!

Make sure you save **paintByNumbers.py** and **pbnFunctions.py** as you will be submitting these files for Lab 06.

Submission

Once you have completed this lab it is time to turn in your work. Please submit the following files to the Lab 06 assignment in Canvas:

- **averages.py**
- **poem.py**
- **paintByNumbers.py, pbnFunctions.py**

Sample Output

averages.py

```
13
42
19
352
17
652
53
84
35
The average of the numbers is 140.77777777777777
```

poem.py

```
Please input the name of the poem you wish summarized: Snowball
Snowball does not exist! Please input the name of the poem you wish
summarized: Snowball.txt
```

In **Output.txt**:

```
The name of the poem is Snowball
The author of the poem is Shel Silverstein
The number of lines in the poem is 8
A preview of the poem is:
I made myself a snowball
As perfect as could be.
I thought I'd keep it as a pet
```

paintByNumbers.py, pbnFunctions.py

```
Please input file you wish to have painted: image
image does not exist! Please input file you wish to have painted:
image.txt
image.txt does not exist! Please input file you wish to have painted:
image.csv
Your image can be found in painting.txt . Enjoy!
```

In **painting.txt**:

```
 ,_,
(O,O)
(   )
-"-"------
```

Rubric

For each program in this lab, you are expected to make a good faith effort on all requested functionality. Each submitted program will receive a score of either 100, 75, 50, or 0 out of 100 based upon how much of the program you attempted, regardless of whether the final output is correct (See table below). The scores for all of your submitted programs for this lab will be averaged together to calculate your grade for this lab. Keep in mind that labs are practice both for the exams in this course and for coding professionally, so make sure you take the assignments seriously.

Score	Attempted Functionality
100	100% of the functionality was attempted
75	<100% and >=75% of the functionality was attempted
50	<75% and >=50% of the functionality was attempted
0	<50% of the functionality was attempted