COS 125 – HMWK3: Lists, Logic, Loops
- Data Analysis & Transformation

You can reference the textbook (in BrightSpace), class materials, and tutoring. Do not use AI. Include the comment header in all homework assignment files, specify any collaboration you had.

Seek help if you get stuck on any aspect of the programming. Boardman 138 has many working there who are happy to help.

Starting an assignment:

- read the assignment when it's posted
- create an outline or flow chart of the inputs, outputs, and processes of the program
    - how do things flow through the program? What do you need from the user? What do you have for constants and outputs? Any special formatting?
    - What process does the data need to go through; just math, logic (if/else), loops, all of the above?
    - go to Boardman for help
    - return to Boardman 138 if you get stuck on some elements

Folder Name

Include all .py files in the folder, then compress and submit the folder with all the files.

yourName_DataTransformer (replace "yourName" with your actual name, e.g., lauraGurney_dataTransformation)

Topics

- Practicing elements and concepts covered in class to demonstrate understanding
- Not required but strongly suggested: plan your program before coding it.
- Simple Python program
- Getting user input
- Converting data types
- Manipulating data in-program
- Implementing decision/control structures and loops
- Implementing and manipulating lists and strings
- Displaying data (output)

Instructions

Only use concepts presented in the course to-date. Using concepts beyond the course does not demonstrate understanding of the concepts presented and may result in a zero for the assignment. Don't forget the comment header with data updated within.

Part 1: yourName_dataAnalyzer.py (30pts)

This program will focus on creating and manipulating a dataset to perform basic analysis.

1. **Dataset Creation:**
    - Ask the user for a positive integer, which will represent the number of data points.
    - Create a list called data_list.
    - Use a loop to prompt the user to enter as many integer values as the user gave as a positive number, one at a time, and add each integer to data_list.
        - What is the most effective method to do this task?
2. **Data Analysis:**
    - Calculate the **sum** and **average** of all the numbers in the data_list.
    - Find the **minimum** and **maximum** values in the list without using built-in functions like min() or max(). You must use a loop and conditional logic.
3. **Data Categorization:**
    - Create two new lists: even_numbers and odd_numbers.
    - Iterate through the data_list and add each number to the appropriate new list based on whether it is even or odd.
    - Count how many numbers are in each new list without using a built-in function – practice looping structures.
4. **Display Results:**
    - Print the original data_list.
    - Print the calculated sum and average.
    - Print the minimum and maximum values.
    - Print the even_numbers and odd_numbers lists, along with their respective counts.

Example Output: (same content, doesn't need to look exactly the same)

```
Enter the number of data points: 5
Enter integer #1: 12
Enter integer #2: 3
Enter integer #3: 25
Enter integer #4: 8
Enter integer #5: 15

Original Data: [12, 3, 25, 8, 15]
Sum: 63
Average: 12.6
```

Minimum Value: 3
Maximum Value: 25

Even Numbers: [12, 8] (Count: 2)
Odd Numbers: [3, 25, 15] (Count: 3)

Part 2: yourName_sentenceGenerator.py (30pts)

This program focuses on string and list manipulation to algorithmically create new sentences.

Practice concepts:

- **String and List Manipulation:** Working with text data, accessing characters, and working with lists, accessing items, indices, and modifying lists.
- **Loops and Iteration:** Repeating a process to generate multiple sentences.
- **Conditional Logic:** Implementing different word choices based on certain criteria.
- **Algorithmic Thinking:** Developing a step-by-step process to transform a sentence in a patterned way.

Problem Description:

Create a "sentence generator," taking a base sentence and algorithmically remixing it. You will write a program that uses a pre-defined set of words to generate a series of modified sentences by systematically replacing words or adding phrases based on a defined pattern.

1. **Define Word Lists:**
    - Ask the user for a list of **nouns** (e.g., ['cat', 'dog', 'book', 'car']).
    - Ask the user for a list of **adjectives** (e.g., ['fluffy', 'fast', 'small', 'bright']).
    - Ask the user for a list of **verbs** (e.g., ['ran', 'jumped', 'slept', 'danced']).
    - Print out the three lists as reference information to the user.
2. **Get a Base Sentence:**
    - Prompt the user to enter a simple, one-word sentence starter for your generated sentences to start with. For example, the user might enter "The."
3. **Implement your Algorithm:**
    - Your program should use a loop to generate five new sentences.
    - For each new sentence, combine the user's sentence starting word with a word from each of the three word lists you created.
    - Use the loop counter (the iteration number) to select the words from your lists. This demonstrates implementation of indexes in lists.
        - For example, in the first iteration, use the first word from each list; in the second iteration, use the second word, and so on.
        - Use the format: [Base Sentence] + [Adjective] + [Noun] + [Verb].

    - Add a period to the end of each generated sentence.
    - Use f-string to format your sentences.
4. **Output Transformed Sentences:**
    - Print each newly generated sentence to the console.

Example Result: (should look similar but does not need to look exactly the same).

After the user enters the required 3 lists of nouns, adjectives, verbs.

Input:

```
Enter a base word for your sentences: Purple
```

Output:

```
Purple fluffy cat ran.
Purple fast dog jumped.
Purple small book slept.
Purple bright car danced.
Purple fluffy cat ran.
```

Grading will be based on:
Implementing course content correctly.
Implementing logic, loops, and data types properly.

Criterion	Exceeds	Meets	Partially Meets	Does not Meet
Interpretation	The code is concise, clear, and efficient. It uses insightful representations to model the problem. The student has a deep and thoughtful understanding of the project and problem requirements.	The student accurately converts the problem's requirements into code. All inputs are correctly interpreted, and the calculations are successful and comprehensive.	The student attempts to model the problem but makes minor errors in their interpretation. For instance, they might correctly model the loop but make a small error in the calculation of the annual rate increase.	The student's code fundamentally misunderstands the problem. They may attempt to use a loop but fail to correctly update variables, leading to a completely inaccurate model of the payback period.
Logical Structures	The program correctly uses logical decision structures (e.g., if/elif/else) to handle all possible scenarios, including edge cases, such as a zero or negative savings percentage. The use of these structures is efficient and well-placed.	The program correctly uses if/elif/else to handle the core logic and ensure the program doesn't crash on invalid input, if applicable.	The program attempts to use conditional logic but makes minor errors that may cause the program to fail under certain conditions.	The program fails to use conditional logic, leading to crashes or incorrect output for different user inputs.
Looping	The student implements the loop	The student correctly	The while loop is either not implemented or	The student fails to implement a while loop

	to complete required processes perfectly. The loop terminates correctly and efficiently, and the variables are updated within the loop correctly.	implements the while loop, and it functions as intended, terminating when the initial cost is covered.	contains logical errors (e.g., infinite loop, incorrect termination condition) that prevent it from functioning properly.	or uses an incorrect loop type.
Data Handling	All inputs are stored in self-documenting variables, and the program demonstrates robust data conversion that handles unexpected user inputs.	The program correctly prompts the user for all inputs, stores them in descriptive variables, and converts data types correctly for calculations.	The program prompts for some but not all inputs, or the variable names are not clear. There may be minor errors in data type conversion that lead to incorrect output.	The program does not prompt for all inputs, or fails to store them correctly. Data type conversion is either not attempted or is consistently incorrect.
Communication & Formatting	The program's output is highly effective and easy to read. It uses excellent formatting (\t, \n) and high-quality explications to present the quantitative evidence, such as the year-by-year savings, clearly and concisely.	The program's output presents the quantitative evidence in a way that supports the conclusion, but the formatting may be less than completely effective. The student correctly formats output according to requirements.	The program's output uses quantitative information but does not effectively connect it to the overall purpose. The output is a series of numbers without a clear explanation of what they mean.	The program fails to provide adequate explicit numerical support. It may use vague, quasi-quantitative words (like "a lot of years") instead of actual numbers, or the output is completely absent or incomprehensible.
File & Folder Organization	The file and folder naming and organization are professional, clear, and logical.	The file and folder naming and organization are correct.	The file or folder naming is slightly off, but the file is still recognizable.	The file and folder naming and organization are incorrect, making the file difficult to find or identify.
Header Comments Zero grade if not included	The student includes a comment header that is well-formatted and includes all of the required information along with a brief description of the program's functionality.		Comment header in present without required information and is significantly lacking relevant information.	A comment header is not included.