ENGR 102 - Lab Assignment #7

Fall 2019

General course learning outcomes:

- demonstrate the use of basic programming techniques in the construction of computer programs, including techniques to collect, store, and manipulate data within a computer program.
- apply programming techniques to solve problems in engineering.
- complete a team programming assignment that ties together concepts learned in the class.

This week is a mixed practice, focusing on looping, but including: user-input/output, conditionals, lists, for-loops and/or while-loops. Do not use SymPy, and do not use "break".

Warm-ups:

Activity 1 *(10-minutes)*:

Create a list of strings, such as: ['arroyo', 'elephantine', 'toy', 'shines']. Write a Python program to count the number of strings where the string length is 2 or more, and the first and last character are the same.

Activity 2 *(10-minutes)*:

Print out all elements of a list with a value less than a user-input number.

Activity 3 *(10-minutes)*:

Accept a sentence as input, and calculate the number of times a certain letter (also user-supplied) is contained in that sentence.

Activity 4 *(10-minutes)*:

Solve a function (*e.g.*, y(x) = sin(x) / (sin(x/10) + x/10) for many different values of x between a user-defined min and max, and store the values in a list. Also, print the maximum value of y(x) for the given range.

Activity 5: The Expandables - to do in lab (team)

☑ *Use for and/or while looping structures in Python*
☑ *Use concept of tolerance in engineering problems*

The Maclaurin series expansion for 1/(1-*x*) on an interval from -1 < *x* < 1 is as follows:

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + x^4 + \cdots + x^n$$

Write Python code which asks for input of a value of *x* on the interval -1 < *x* < 1, and which computes an approximation to 1/(1-*x*) using the using the series expansion summation. The summation should be continued until the term to be added to the summation is less than 10^{-6} in absolute value. Hint: Note that every term in the series is x raised to a power (i.e., the first two terms are simply $x^0=1$ and $x^1=x$).

(continued, next page)

Activity 6: Rise of the Widget - to do in lab (team)

☑ *Store and write lists in Python, and loop through values.*

You manage a plant, and as a part of your duties, you measure production each day in number of widgets. You want a program where you can enter the number of widgets produced daily for an arbitrary number of days, and have reported whether production is rising or falling over various periods. Specifically, you would like the report to include ranges from 1-day intervals up to the maximum possible interval (determined from the user input).

For example, if the widget production was entered for 5 consecutive days, and the production levels were 13, 15, 17, 15, 18, you might output something like:

> For 1-day intervals 75.0% were increasing and 25.0% were decreasing
>
> For 2-day intervals 66.7% were increasing and 0.0% were decreasing
>
> For 3-day intervals 100.0% were increasing and 0.0% were decreasing
>
> For 4-day intervals 100.0% were increasing and 0.0% were decreasing

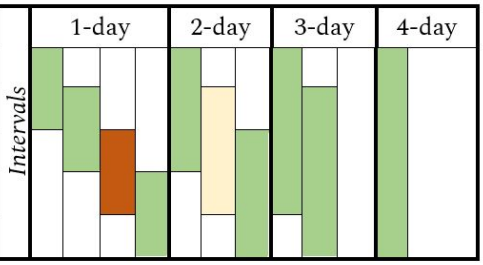
The first line reports the four one-day intervals, where Day 1-2 (13-15), Day 2-3 (15-17) and Day 4-5 (15-18) were increasing, and Day 3-4 (17-15) was decreasing.

The second line reports the three two-day intervals, where Day 1-3 (13-17) and Day 3-5 (17-18) were increasing, while Day 2-4 (15-15) was neither increasing nor decreasing.

The third and fourth lines report that both four-day intervals, Day 1-4 and Day 2-5 were increasing, and the maximum interval (Day 1-5) was increasing.

I've included a chart below to help you visualize what is being reported. The number of days of production input will affect the number of intervals to be calculated and the possible interval lengths.

Day	Production
1	13
2	15
3	17
4	15
5	18



A. Remember to plan! Consider exactly how you will make these computations. The looping in this problem is trickier than what you have encountered previously, and formatting the output may be a challenge for you. Remember to use the methods we have discussed for testing, and using incremental development.

B. Take input for an arbitrary number of days, and stop when the user enters a negative number. Be descriptive.

C. Your program output should report, for each possible interval, from 1 day to the maximum, what percentage of intervals had increasing production and what percentage had decreasing production. *Print the output with 1 digit after the decimal.*

There is more than one way to get one decimal place after the number. One option is for your team to look up and learn the command for formatting floating-point output *(zyBook 3.9 may be helpful here)*. Another option is for your team to come up with a set of steps to create this string, yourselves.

(continued, next page)

Activity 7: Making the Cut in Golf - to do in lab (team)

☑ *Create appropriate list of variables, and plan how to create a Python program.*
☑ *Create Python program to read user input, perform necessary data reformatting, and print the expected output.*

Professional golf tournaments typically last four rounds, and a player's score is the sum of their individual round scores (the lower the better). It is common for all players to play the first two rounds of golf. A "cut" score is determined, and those whose scores are better (*i.e. lower*) than the cut are allowed to play the remaining rounds of the tournament (they "made the cut") while the rest do not.

a. Write a program that reads in an arbitrary number of golfers' names, and their first and second round scores. Specifically, it should read the first round score on one line, then the second round score on another line, then the player's name on a third line. The user should indicate they are done entering players by giving a negative score for the first round (the reading of data should stop without attempting to read a second round or player name when this occurs).

b. The cut for our tournament will be the median score among the golfers. *You are not to use the built-in command to find the median or sort the data.*

   There are many ways to find a median. A major part of this problem is to figure out a method for finding the median, yourselves. There are solutions involving multiple loops, there are solutions using multiple lists, and so on. Commonly the numbers are sorted from smallest to largest, and then the median is found or calculated directly.

c. Print out the names of golfers who made the cut and those who did not make the cut. You will be outputting two sets of names; be clear which is which.

Do this project as a team. The idea is that you should talk through the problem and develop a solution together. Be sure to use good code development, as discussed before.

(continued, next page)

Additional Practice *(Optional)*

AP-1:

Square each even number in a list, and cube each odd number.

AP-2:

Write a Python program to insert an element before every element in a list.

AP-3:

Insert an even number before every odd number in a list—but only if the number before the odd number isn't already even.

AP-4:

Write a Python program to print the numbers of a specified list after removing even numbers from it.

AP-5:

Check whether a list contains a specific sublist.

```
e.g., list_a= [1, 2, 2, 5, 3, 2], sub_list=[2,3]  → False
e.g., list_a= [1, 2, 2, 5, 3, 2], sub_list=[5,3]  → True
```

AP-6:

Remove duplicates from a list.

```
e.g., [1, 2, 2, 5, 3, 2]  → [1, 2, 5, 3]
```

AP-7: Cows and Bulls

Randomly generate a 4-digit number. Ask the user to guess a 4-digit number. For every digit that the user guessed correctly in the correct place, they have a "cow". For every digit the user guessed correctly in the wrong place is a "bull." Every time the user makes a guess, tell them how many "cows" and "bulls" they have. Once the user guesses the correct number, the game is over. Keep track of the number of guesses the user makes throughout the game and tell the user at the end.

Say the number generated by the computer is 1038. An example interaction could look like this:

```
Welcome to the Cows and Bulls Game!
Enter a number:
 >>> 1234
 2 cows, 0 bulls
 >>> 1356
 1 cow, 1 bull
 ...
```

Until the user guesses the number.