ENGR 102 - Lab and Assignment #7

Activity #0: Lecture Review (Group)

This activity is to provide practice with using lists. You should work as a team to discuss how the commands work. It would be best to have everyone make a guess before executing the code. If you guess correctly, awesome. If not, try some other similar commands and work out what is happening.

Create the following lists:
A = [10, 87, 101, 1, 43, 7, 8, 15, 123, 95, 77, 10]
B = [‘cable’, ’engr102’, ‘Zachry’, ‘INSTRUCTS’, ‘in’, ‘students’, ‘with the’, ‘Best’]
C = [[1, 2, 3], ‘proton’, ‘electron’, [[‘A’, ‘B’],[1, 2, 3, 4], 5, ‘cable’], [‘a’, ‘b’, ‘c’, ‘d’, ‘e’], ’f’]

Try the following:
Indexing and Slicing:

i. A[:], A[5], A[3], A[-1], and A[-3]
ii. B[:4], B[4:], B[0:], B[:2], B[1:-1], B[3:-2], and B[:5:-3]
iii. C[0][1], C[3][0][:], C[3][1][:2], C[3][1][:-3], C[:], C[3][1][::-1], and

Problems:

i. Find and print the median and mean of A without using median or mean functions.
ii. Find and print the min and max of A without using min/max functions.
iii. Write a program to sort A in descending order without using the sort function.
iv. Create a new list AA that contains only the odd values of A.
v. Find and print the average number of characters for the elements of B.
vi. Using all of the elements of B, construct and print a sentence.
vii. Sort the elements of C in order of increasing length.
viii. Find and print the letter that occurs most frequently in B.

Activity #1: Comparing array values (Group)

This activity is meant to help you learn to read data, store it in a list, and then process it by looping through values.

Imagine that you are managing a plant, and measure production each day. Production is in number of widgets, and you want a program that will let you enter in the number of widgets, and then tell whether for periods of time ranging from 1 day intervals to the maximum interval, whether the production is rising or falling. You want this to work for an arbitrary number of days. That is, you should be able to enter as many days as you want, and get a report as to days increasing or decreasing.

For the input, make sure your program can collect the number of widgets for an arbitrary number of days. You can stop taking input when a user enters a negative number. Be sure to give a descriptive message telling what to input.

For the output, you want to report, for each possible interval from 1 day to the maximum interval, what percentage of differences in widget production were increasing, and what percentage of intervals were decreasing (some could be the same). *Print the output with 1 digit after the decimal.*

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 is because out of the four one-day intervals, 13-15, 15-17, and 15-18 were increasing, while 17-15 was decreasing. The second line is because out of the three two-day intervals, 13-17 and 17-18 were increasing, while 15-15 was neither increasing nor decreasing. The third line is because both four-day intervals, 13-15 and 15-18, were increasing, and the final line is because the one four-day interval 13-18 was increasing.

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. Another option is for your team to come up with a set of steps to create this string, yourselves.

Before your team starts coding, 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 (work together as a team to figure out how you might print the percentages with exactly 1 place after the decimal). Remember to use the methods we have discussed for testing, and using incremental development.

Activity #2: Making the cut in golf (Group)

The purpose of this activity is to get your team, as a team, to think through a computational problem and come up with an interesting computational solution. There are multiple possible ways to solve this problem.

In professional golf tournaments, a tournament typically lasts four rounds. A player's score is the sum of the scores in the rounds (the lower the better). It is common for all players to play the first two rounds of golf. However, only a fraction of the players are allowed to continue on to the third and fourth rounds. The way this is done is that a "cut" score is determined, and those whose scores are better (i.e. lower) than the cut are allowed to play the remainder of the tournament (they "made the cut") while the rest do not.

In this case, the cut will be the median score among the golfers. You are to print out the names of golfers who made the cut (whose score was below the median), and those who did not make the cut (score above the median). Thus, about half the golfers should make the cut. Here are some details:

- You are to write a program that reads in golfers' names, and their first and second round scores. Specifically, they should enter the first round score on one line, then the second round score on another line, then the player's name on a third line. This should continue for an arbitrary number of players. The user should indicate that they are done entering players by giving a negative score for the first round – at this point, the reading-in of data should stop (without reading a second round or player name).

- In practice, a common way to find the median is typically to sort the numbers from smallest to largest, and then to find the median from there directly (either the middle element if an odd number of elements., or the average of the two middle elements if an even number of elements). There is a simple sort routine built in to Python, ***but you are not to use it (the built-in sort command) for this problem***.
- 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 just multiple loops, there are solutions that will use more than one list, and so on. Also, you do not technically have to find a median – you could just find a way to report both those above and those below the median.
- You'll be outputting two sets of names. Be sure it is 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.

Activity #3: Chessboard moves (Group)

The purpose of this activity is to get you used to using lists of lists, in a 2-D matrix-like format.

Write a program that sets up a chess board with chess pieces, and lets people make moves (of one piece at a time) on the chess board. Here are the details:

- The chessboard is an 8x8 board.
- Display the chess board before every move. Each empty square should just be a period. Each square with a piece should have the piece's identifier.
- For identifiers, use lower-case for the white pieces, upper-case for the black pieces.
    - Use P/p for pawn, R/r for rook, N/n for knight, B/b for bishop, Q/q for queen, and K/k for king.
- When a piece is moved, the location it moves from is then an empty square, and it then occupies the square it is moving to. If there was already a piece in the square it is moving to, then that other piece is eliminated from the game.
- **Important:** You do **NOT** need to enforce any rules of chess or verify moves, with one exception:
    - If someone puts in a move from a position where there is not a piece, report an error and exit the program.
    - Other than that, you don't need to worry about the sides alternating turns, about pieces moving in ways they aren't allowed, about landing on your own pieces, etc.
    - Only one piece moves at a time, and it can move to any position on the board.
- You can specify the system you want the user to use to specify the starting and ending location.
    - In reality, chess uses a standard system, where the columns are labeled a-h from left to right and the rows are labeled 1-8 from bottom to top. You do not have to use this system, but can if you wish.

As an example, here is what the board would look like at the very beginning of the game:

```
RNBQKBNR
PPPPPPPP
........
........
........
```

```
........
pppppppp
rnbqkbnr
```

Remember to discuss this as a team and think through exactly what you will do before developing it!