ENGR 102 - Lab Assignment #6B

Fall 2020

You are to write the following program as described below. For all programs, include comments in your code that describe the purpose of individual blocks. Remember the appropriate header information.

Program 1: Miss a Moss Mass

☑ *Create lists in Python*
☑ *Perform operations on lists*

For one month you took daily measurements of the growth of a particular type of moss. The recorded mass measurements (in *g*) are listed here:

```
[15.80, 19.60, 21.85, 33.61, 49.73, 51.27, 56.26, 63.06, 76.56, 76.57, 85.78, 90.74, 92.60, 99.71,
100.51, 101.12, 101.25, 102.19, 104.85, 110.59, 125.92, 131.25, 136.04, 141.15, 148.54, 150.02]
```

a. You realize the last recorded mass was omitted from the list. Append the value 162.76*g* to the list.
b. You notice that the data is missing the 9th day of data, with a value of 71.01*g*. Insert this into the data set list in the appropriate location.
c. Have your program print out how many days of data were taken.
d. Have your program find the average value of moss mass during each week of data. Begin by creating a new list for each week of data as a slice of the original list.

   (Hint: look up `mean()` *for Python)*
e. What is the daily growth rate (*g/day*) of the moss for week 1, week 2, week 3, week 4, and for the entire month?
f. Format your output to look approximately as shown below:

```
There were 28 days of data measurements taken.
--------------------------------------------------
Average moss mass in week 1:        35.45 g
Average moss mass in week 2:        79.47 g
Average moss mass in week 3:       102.89 g
Average moss mass in week 4:       142.24 g
--------------------------------------------------
Average daily growth during week 1:  6.75 g/day
Average daily growth during week 2:  5.24 g/day
Average daily growth during week 3:  3.74 g/day
Average daily growth during week 4:  5.26 g/day
--------------------------------------------------
Average growth for the month:             g/day
```

(continued, next page)

Program 2: Evenly Upward

☑ *Use of while and for looping structures*
☑ *Creating and modifying lists*

Read in user-input integers one at a time until the user types 'q', storing all inputs in a list. Then, print out the even numbers in increasing order, from smallest value to largest.

Program 3: Collatz Conjecture

☑ *Create a list using a looping structure.*
☑ *Create plots using the MatPlotLib package*

The Collatz conjecture, also known as the 3n+1 conjecture (and other names), posits that the following sequence of numbers always eventually reaches 1: Given a number n, if n is even then the next number is n divided by 2; if n is odd, then the next number is 3n+1. As simple as this seems, it is unproven (and considered extremely hard to prove) by mathematicians.

a. Create a list that starts at a user-specified positive integer, and appends subsequent values as calculated by the Collatz sequence above. Create a second list that starts at 0 for the user-input value, and appends the counting value for each step taken.

   As an example, if the user inputs 6, you will have the following two lists created:

```
List_1: [6, 3, 10, 5, 16, 8, 4, 2, 1]   (sequence values)
List_2: [0, 1, 2, 3, 4, 5, 6, 7, 8]     (counting numbers)
```

b. Create a **plot** of the Collatz sequence you generated vs. the count of steps taken.

Program 4: Pa$$w0rd Pr0t3ct1oN

☑ *Use of dictionaries in Python*

Write a single program that *first* reads from a user a set of multiple usernames and matching passwords, and *then* simulates a user log-in. Use dictionaries for this exercise.

Program 'Phase 1': For the first aspect of the program, a) read in an integer that states the number of username/password pairs that will be typed in next; b) read in a set of user names, one per line; and c) read in a set of passwords, one per line.

For example, input might be:

```
2
John
Joe
SecreT_passWorD
G$a-4(ztY
```

Program 'Phase 2': Once these are input, prompt the user for a username followed by a password. Your program should repeatedly ask the user to type in a username followed by a password. If they input a valid username and password combination, print a message that they are allowed into the system. If they have an incorrect username/password, show an error and allow them to try again (repeatedly).

Note: passwords should never really be stored unencrypted within a program like this.