ENGR 102 - Lab Assignment #6

Fall 2020

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; and collect, create, store and manipulate data in larger structures such as lists.
- apply programming techniques to solve problems in engineering, including plotting data.
- complete a team programming assignment that ties together concepts learned in the class.

Activity 1: Gotta Catch One of 'Em!

☑ *Use of while looping structure*
☑ *Creating lists and using list operations*

Create a program to keep track of your most awesome Pokémon! The program should initially ask for a Pokémon name, current Combat Points (CP), current Level (1–40), and current number of Candies. The program should then print the following menu:

```
<Pokemon Name>
Current CP:     <CP value>
Current Level:  <Level value>
Candies:        <Candy value>
1 - Add Candy
2 - Use Candy to Level-Up
3 - Exit to Main Menu
```

- Use a single *list* to store all information regarding the Pokémon; do not create separate variables for CP, Level or Candies. Use integers for all number values.
- Selecting "Add Candy" will ask the user how many candies to add, and then update the value of Candies by the correct amount.
- Selecting "Use Candy to Level-Up" will use 1–2 Candies collected to level up the Pokémon one level.
    - Each candy will level the Pokémon by 1 level from levels 1–30,
    - It requires 2 candies to level-up to levels 31–40.
- The CP increase is calculated as CP * 0.0094 / [0.095 * sqrt(Current Level)]^2 for levels 1-30.
- The CP increase is calculated as CP * 0.0045 / [0.095 * sqrt(Current Level)]^2 for levels 31-40
- Level 40 is the maximum level
- Updated information should be shown to the user after every updated value.

(continued, next page)

Activity 2: Going Places - to do in lab (team)

☑ *Create lists using a looping structure.*
☑ *Use list operations on existing lists.*
☑ *Create plots using the MatPlotLib package*

A traveling Aggie is about to head out, to visit some friends. Our Ag wants to do some fancy calculations this time, and creates a painfully created fuel economy model:

$$MPG = -5.9852 + 1.6052*V_{MPH} - 0.0141*(V_{MPH})^2$$

The result (MPG) is a value in miles traveled per gallon of fuel, where the input V_{MPH} is the car velocity in miles per hour (MPH).

Since our Aggie isn't entirely certain who of the many friends to visit yet, there needs to be a user input for the distance to travel.

Part 1: Creating the Data

Realizing that speed limits are suggestions, and that you can go under (but never over) the posted limit, means there are many speeds that are possible to travel. Our Ag wants to know more information about how different speeds affect the fuel economy, time of travel, and the cost of a trip.

a. Take user input of current fuel cost ($/gal), and the distance of the proposed trip.
b. For speeds ranging from 5 to 80 mph, in 5 mph increments, create four lists to store MPH, MPG, cost, and time of travel.

Part 2: Using the Data

Create a menu (using print statements, conditional statements, and looping) for our friendly classmate, giving the following options and outputs.

1 – Create a table displaying MPH, MPG, Cost, and Time of the travel.
2 – Create a plot from 25–80 mph vs. cost. For this plot, let's add labels. *Hint: use the* `xlabel('')` *and* `ylabel('')` *functions available in the matplotlib.pyplot module.*
3 – Create a plot of cost vs. time of travel for speed values between 25–80 mph. Add labels.
4 – Calculate the cost and time at a specific speed.
5 – *(Challenge Activity–optional, for glory)* Calculate the required minimum cost and speed to make the distance in a user-specified time (and give a warning if the time limit can't be met within the range of tabulated speeds).
6 – Quit

If the user types a value other than an integer 1–6, the program should prompt the user to input a valid selection, and take a new input. After calculating item 1, 2, 3, 4 or (optionally) 5, the program should then take a new user selection from 1–6, until eventually the user selects 6 to quit the program.