Lab 06: Lists

Goals for this lab:

1. Use a single dimensional `List` to store data
2. Add, remove, and search for data in a `List`
3. Use for loops to allow the program to iterate through a `List`
4. Use a multi-dimensional, or nested, `List` to store data
5. Use the matplotlib.pyplot module to plot a line graph using `Lists`
6. Write and execute simple Python programs using the given instructions

General Guidelines:

- Use meaningful variable names
- Use appropriate indentation
- Use comments, especially for the header, variables, and blocks of code.

Example Header:

```
# Author: Your name
# Date: Today's date
# Description: A small description in your own words that describes what the program does. Any additional information required for the program to execute should also be provided.
```

Example Function Docstring:

```
"""
General function description
:param p1: DESCRIPTION
:type p1: TYPE
:param p2: DESCRIPTION
:type p2: TYPE
:return: DESCRIPTION.
"""
```

1. Preparation

You will be using one premade Python file for this lab, so be sure to download **trebuchet.py** from the Canvas Lab 05 link before you begin the lab. **NOTE:** Be sure to download the file from Canvas, not copy and paste the file's contents to avoid issues with the code.

2. Farther and Faster

In the previous lab, you used six variables to store the first, second, and third furthest distances achieved by the trebuchet. However, data of this nature can be stored in a `List` to make it easier to access and alter. For this lab you will be updating the **trebuchet.py** file to utilize two instances of a `List` rather than the six variables, and the file should be renamed to **trebuchetUpdate.py**.

Your program should now perform the following steps:

- Create a `List` to store the three distances, such that each distance is initialized to 0
- Create a `List` to store the trial numbers, such that each trial number is initialized to 0
- Update the remainder of the program such the distance and the trial numbers, are now only stored in the two `List` data structures
    - Remember that in a `List` the first position is always index 0

Make sure you save your **trebuchetUpdate.py** as you will be submitting this file for Lab 06.

3. Lab Manager

While much of physics can be very theoretical and modeled with large series of complex equations, certain aspects can be quite practical and experimented on with the right equipment, assuming that you have that equipment!

For this part of the lab, you will need to manage a small laboratory's equipment inventory in your **labManager.py** . The laboratory only has enough space and funding to store a maximum of five pieces of equipment at a time and should allow the user to add equipment up to that maximum, remove equipment currently owned by the laboratory, and display what is currently owned.

Your program should perform the following steps:

- Your program should have a `List` representing the laboratory and all of the equipment contained within. The `List` should start out empty.
- Your program should display a numbered menu with the following options:
    - 1. Add Equipment
    - 2. Remove Equipment
    - 3. Display Current Equipment
    - 4. Leave the Laboratory Manager
- You should allow the repeatedly choose which of the four actions they wish to perform until they choose to leave
    - Make sure you redisplay the menu after each action

- If the user wishes to add equipment
    - Check to make sure adding the equipment would not exceed the five-item limit
        - If it would exceed the limit, inform the user they cannot add equipment with an appropriate message
        - Otherwise prompt the user for what they would like to add and append the equipment to the end of the `List` using the appropriate `List` function, and inform the user the equipment has been added with an appropriate message
- If the user wishes to remove equipment
    - Iterate through the `List`, and if the equipment is present in the `List`, remove it using the appropriate `List` function, and inform the user of the removal with an appropriate message
        - You may want to use a variable to keep track of if the equipment was found or not
    - If the equipment is not present in the `List`, inform the user of this with an appropriate message and do not remove anything
        - Be careful as using the `remove()` function will crash the program if the specified element is not in the `List`, and we do not want that to happen!
- If the user wishes to display what equipment is currently owned by the laboratory
    - Iterate through the `List` and output each piece of equipment in a visually pleasing format
    - If you wish to have multiple outputs print on the same line you can add `, end=" "` to add a space at the end of the output rather than a newline, e.g. `print("Item",end=" ")`
- If the user wishes to leave the laboratory manager
    - Wish them well on their journey of discovery with an appropriate message
- If the user chooses an option not in the menu inform them of their mistake

Make sure you save your **labManager.py** as you will be submitting this file for Lab 06.

4. Cosmic Rays

"Cosmic rays are energetic charged particles passing through Earth's atmosphere, originating from our galaxy and thought to be produced by events such as supernovae or black holes. They are composed primarily of protons but also of helium nuclei as well as a small portion of heavy nuclei. When these cosmic rays interact with the atmosphere, they produce showers of new particles, which can decay in many different ways. In particular, pions and kaons produced in this manner decay to muons at an altitude of about 15,000 meters. These muons then travel in conical showers within an angle of 1◦ of the original trajectory of the particles which produced them. Muons have a lifetime of 2.2 µs, after which they decay into an electron, a neutrino and an antineutrino. Without taking into account relativity, a typical muon would only travel approximately 660 m before decaying. However, because of time dilation in the rest frame of the muon, as well as its relatively weak interaction with other particles, it actually travels much farther and can be detected at Earth's surface. Since most other particles are completely stopped when passing through the atmosphere, cosmic ray showers observed at sea level are almost exclusively composed of muons. The muon flux at sea level is known to be approximately 1 per minute per square centimeter per steradian."
(https://gustavus.edu/physics/concertFiles/media/Cosmic_Ray_Muon_Detection_Thesis.pdf)

A cosmic ray muon detector (CRMD) is a particle detector that can capture and detect these particles. For this program, you will be simulating an array of CRMD's on a two-dimensional map, then finding the location with the highest capture rate and the location with the lowest capture rate and outputting the entire map as a grid. Your program should be saved in a file named **muons.py** .

Your program should perform the following steps:

- Create a constant variable to store the dimensions of the square map and assign it the value 8
- Create an 8x8 two-dimensional `List` of integers with the value 0
    - You will need nested `for` loops for this
- Iterate over the entire two-dimensional `List` and for each slot assign a random value in the range of 0-500, inclusively
- Create two variables to store the x and y coordinates of the highest capture rate
- Create two variables to store the x and y coordinates of the lowest capture rate
- Create one variable to store the value of the highest capture rate, and initialize it to an appropriate value
- Create one variable to store the value of the lowest capture rate, and initialize it to an appropriate value
- Iterate over the entire two-dimensional `List` and for each slot:
    - Determine if it is the highest known capture rate on the map, and if it is, update the appropriate variables
    - Determine if it is the lowest known capture rate on the map, and if it is, update the appropriate variables
    - In the event of multiple highest/lowest capture rate, use the first one discovered
- Inform the user the value of the highest capture rate on the map and its x,y coordinates, using an appropriate message
    - You should use human counting for the x,y coordinates and start at 1 instead of 0
- Inform the user the value of the lowest capture rate on the map and its x,y coordinates, using an appropriate message
    - You should use human counting for the x,y coordinates and start at 1 instead of 0
- Print out the two-dimensional `List` in an easy-to-read grid format
    - If you wish to have multiple outputs print on the same line you can add `, end=" "` to add a space at the end of the output rather than a newline, e.g. `print(val,end=" ")`

Make sure you save your **muons.py** as you will be submitting this file for Lab 06.

5. matplotlib

Part of the great strength of Python comes from the many libraries, or modules, that are available to provide a myriad of different functionalities. From machine learning to parallel processing, data analytics to bioinformatics, numerous modules, both for free and at a cost, can be downloaded and used in your programs.

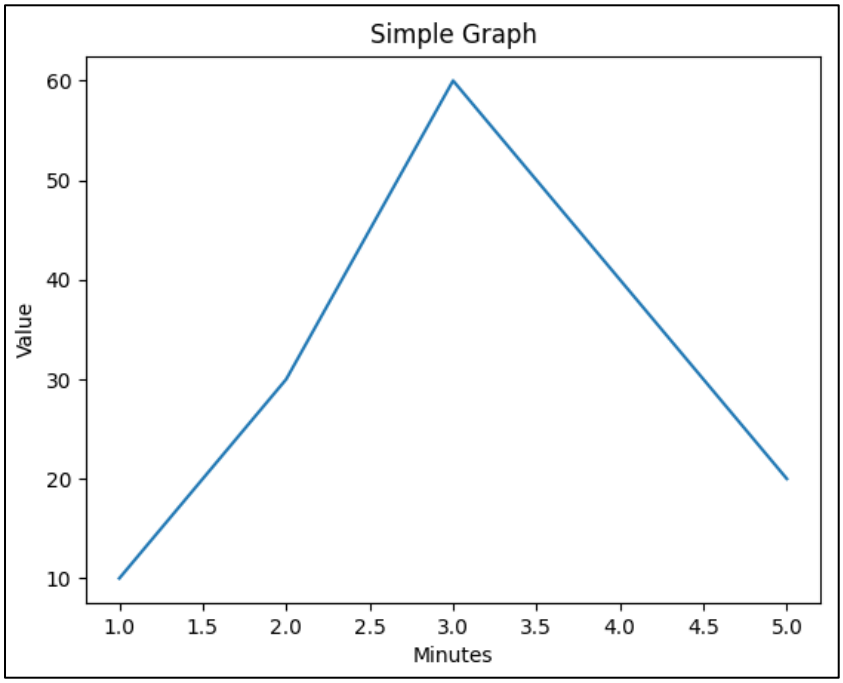
To install a new Python module, we can use the Python package manager, pip, and a few simple commands.

1. On Windows, open the Command Prompt or Powershell/On MacOS or Linus, open the Terminal
2. Once opened, type in `pip install matplotlib` and press enter
3. Your screen should look something like the following:

```
C:\Users\josep>pip install matplotlib
Collecting matplotlib
  Downloading matplotlib-3.8.0-cp311-cp311-win_amd64.whl (7.6 MB)
     ---------------------------------------- 7.6/7.6 MB 48.9 MB/s eta 0:00:00
Requirement already satisfied: contourpy>=1.0.1 in c:\users\josep\appdata\local
ges (from matplotlib) (1.1.0)
Requirement already satisfied: cycler>=0.10 in c:\users\josep\appdata\local\pro
(from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\josep\appdata\loca
ages (from matplotlib) (4.40.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\josep\appdata\loca
ages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy<2,>=1.21 in c:\users\josep\appdata\local\p
s (from matplotlib) (1.25.0)
Requirement already satisfied: packaging>=20.0 in c:\users\josep\appdata\local\
es (from matplotlib) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\josep\appdata\local\pr
 (from matplotlib) (10.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\josep\appdata\local
ges (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\josep\appdata\l
ackages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\josep\appdata\local\program
m python-dateutil>=2.7->matplotlib) (1.16.0)
Installing collected packages: matplotlib
Successfully installed matplotlib-3.8.0
```

4. If the system prompts you to install any additional dependencies, choose yes, typically by typing Y and pressing enter.

Matplotlib should now be installed on your system, so all we need to do is test it out! Using the provided **simpleGraph.py** try running the program and see if a line graph appears! It should look something like the following:



If you are having errors, please email your instructor or TA to help, or speak with your classmates to see if perhaps they have some suggestions as well.

A few things to note in this program, first we need to create one `List` for the x axis and one `List` for the y axis. Then we call the `plot()` function and provide it the `List` for the x axis and the `List` for the y axis. The `title()` function applies a title to the top of the screen, and the `xlabel()` and `ylabel()` add labels to the x and y axis, respectively. Finally, once all of the data has been added and the graph configured, we call the `show()` function to display the graph.

Additionally, you probably also noticed that when we imported matplotlib, we said `import matplotlib.pyplot as plt`. In this case, we are using the `pyplot` submodule to provide our plotting functionality. The reason we add the `as plt` to the end is to simplify our code slightly, because if we did not have that we would have to add `matplotlib.pyplot` in front of every `pyplot` function call! `as someName` allows us to create an alias for a module to reduce the length of the name when we're calling functions from the module. Using an alias is not required, but it does make the code easier to read.

Now, for this part of the lab, create a new file called **tempGraph.py** . This program will generate random temperatures for a series of cities, store those temperatures in lists and create a graph using those lists.

- Make sure you import the `pyplot` submodule and consider using an alias, though make sure it is unique and not the name of a variable in your program!
- Create a new `List` named `time` that stores the integer values 1 through 12, inclusively
- Choose three city names, and for each city
    - Create an empty `List`
    - Append 12 random integer values in the range of 10 through 30, inclusively to the `List`
    - Call the plot() function passing in `time` and your city's `List`
- Provide a relevant title to your graph
- Provide relevant labels to your x and y axis
- Create a legend using the `legend()` function, by passing in a `List` containing the string names of your cities, in the order they have been plotted (i.e., the order that you called the `plot()` functions)
- Call the `show()` function

Your graph should look something like the one in the Sample Output for this program, though the lines will probably be different, and the legend may be in a different place. Matplotlib tries to position the legend automatically so that it covers less of the data in the graph.

Matplotlib contains a considerable amount of functionality and can generate many different plots such as bar plots, scatter plots, pie charts, and more. It is recommended that you go explore what is possible using the geeksforgeeks Matplotlib Tutorial, or check out Matplotlib's Pyplot tutorial!

Make sure you save your **tempGraph.py** as you will be submitting this file for Lab 06.

Submission

Once you have completed this lab it is time to turn in your work. Please submit the following four files to the Lab 06 assignment in Canvas:

- **trebuchetUpdate.py**
- **labManager.py**
- **muons.py**
- **tempGraph.py**

After you have submitted your files, you are welcome to practice some additional coding and explore Spyder.

Sample Output

trebuchetUpdate.py

```
Please enter your distance for trial 1: 298
Would you like to input another trial? (Y/N):Y
Please enter your distance for trial 2: 305
Would you like to input another trial? (Y/N):Y
Please enter your distance for trial 3: 301
Would you like to input another trial? (Y/N):Y
Please enter your distance for trial 4: 312
Would you like to input another trial? (Y/N):Y
Please enter your distance for trial 5: 307
Would you like to input another trial? (Y/N):N
The top three distances for the trebuchet are:
Trial Distance
    4      312
    5      307
    2      305
```

labManager.py

```
Welcome to the inventory manager for your laboratory!

You can choose from the following options:
1. Add Equipment
2. Remove Equipment
3. Display Current Equipment
4. Leave the Laboratory Manager
What would you like to do: 1
What would you like to add to the laboratory: Telescope
Telescope has been added

You can choose from the following options:
1. Add Equipment
2. Remove Equipment
3. Display Current Equipment
4. Leave the Laboratory Manager
What would you like to do: 1
What would you like to add to the laboratory: Electron Microscope
Electron Microscope has been added
```

You can choose from the following options:
1. Add Equipment
2. Remove Equipment
3. Display Current Equipment
4. Leave the Laboratory Manager
What would you like to do: **1**
What would you like to add to the laboratory: **Electromagnets**
Electromagnets has been added

You can choose from the following options:
1. Add Equipment
2. Remove Equipment
3. Display Current Equipment
4. Leave the Laboratory Manager
What would you like to do: **1**
What would you like to add to the laboratory: **Liquid Helium**
Liquid Helium has been added

You can choose from the following options:
1. Add Equipment
2. Remove Equipment
3. Display Current Equipment
4. Leave the Laboratory Manager
What would you like to do: **1**
What would you like to add to the laboratory: **Lenses**
Lenses has been added

You can choose from the following options:
1. Add Equipment
2. Remove Equipment
3. Display Current Equipment
4. Leave the Laboratory Manager
What would you like to do: **1**
Your laboratory cannot support any more equipment!

You can choose from the following options:
1. Add Equipment
2. Remove Equipment
3. Display Current Equipment
4. Leave the Laboratory Manager
What would you like to do: **2**
What would you like to remove from the laboratory: **Telescope**
Telescope has been removed

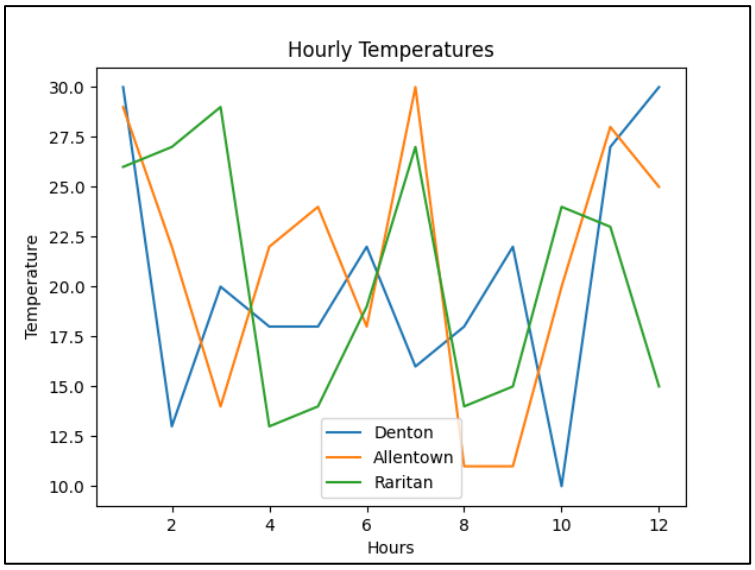
You can choose from the following options:
1. Add Equipment
2. Remove Equipment
3. Display Current Equipment
4. Leave the Laboratory Manager
What would you like to do: **2**
What would you like to remove from the laboratory: **Telescope**

```
Telescope was not present and could not be removed

You can choose from the following options:
1. Add Equipment
2. Remove Equipment
3. Display Current Equipment
4. Leave the Laboratory Manager
What would you like to do: 3
Your laboratory currently contains: Electron Microscope Electromagnets
Liquid Helium Lenses

You can choose from the following options:
1. Add Equipment
2. Remove Equipment
3. Display Current Equipment
4. Leave the Laboratory Manager
What would you like to do: 5
5 was not a valid option. Please try again

You can choose from the following options:
1. Add Equipment
2. Remove Equipment
3. Display Current Equipment
4. Leave the Laboratory Manager
What would you like to do: 4
Good luck on your journey of discovery!
```

muons.py

```
The highest capture rate was 488 at location 4,8.
The lowest capture rate was 25 at location 4,7.
The map looks like the following:
368  346   48  420  353  321   62  478
 91  449  340  447  224  312  428  284
406   61  252  141  462  483  124  142
123   45  361  198   63  384  275  354
131  271   51  270  236  184   79  423
 51  374  351  303  246  354  439  286
236  359   79   25  191  414  234  116
107  101   81  488  133  266  481  105
```

tempGraph.py



Rubric

For each program in this lab, you are expected to make a good faith effort on all requested functionality. Each submitted program will receive a score of either 100, 75, 50, or 0 out of 100 based upon how much of the program you attempted, regardless of whether the final output is correct (See table below). The scores for all of your submitted programs for this lab will be averaged together to calculate your grade for this lab. Keep in mind that labs are practice both for the exams in this course and for coding professionally, so make sure you take the assignments seriously.

Score	Attempted Functionality
100	100% of the functionality was attempted
75	<100% and >=75% of the functionality was attempted
50	<75% and >=50% of the functionality was attempted
0	<50% of the functionality was attempted