Name: Garrett DuBow	**Lab Time** F 1400

Names of people you worked with:

-

Websites you used:

- https://stackoverflow.com/questions/903853/how-do-you-extract-a-column-from-a-multi-dimensional-array
- https://www.youtube.com/watch?time_continue=567&v=W0EPjfz_WVU&feature=emb_logo
- https://kite.com/python/docs/matplotlib.pyplot.xlim
- https://www.geeksforgeeks.org/python-dictionary-keys-method/
- https://pymotw.com/2/collections/counter.html
- https://www.w3schools.com/python/ref_func_range.asp

Approximately how many hours did it take you to complete this assignment (to nearest whole number)?	3+2+3+

The Rules: Everything you do for this lab should be your own work. Don't look up the answers on the web, or copy them from any other source. You can look up general information about Python on the web, but no copying code you find there. Read the code, close the browser, then write your own code.

By writing or typing your name below you affirm that all of the work contained herein is your own, and was not copied or copied and altered.

Garrett DuBow

Note: Failure to sign this page will result in a 50 percent penalty. Failure to list people you worked with may result in no grade for this lab. Failure to fill out hours approximation will result in a 10-percent penalty.

BEFORE YOU BEGIN

1. Download `msd.py` and `counter.py` from Canvas and put in your project
2. Install the Python packages scipy and matplotlib if you don't have them already.
    a. If you're using a Mac, you might have to put these lines at the top of your `lab4.py` file to make things work properly, depending on how you installed Python.

    ```
    import matplotlib as mpl
    mpl.use('TkAgg')
    ```

    b. You might have to install some extra Python packages for this lab (matplotlib and its dependencies). If you don't know how to do this, ask the TA in the lab session for help.
3. Check out the plotting folder in github for examples
4. Work through the PyPlot tutorial

Learning Objectives:

You should be able to do the following:

- Plot and graph data using matplotlib
- Analyze your code for time complexity – O(n), etc

Thoughts (come back and read these after you've read the problems):

1. The `random` module in Python provides many utilities for generating numbers. Be sure to read *and* verify the behavior of the functions you're using, e.g. some integer generation functions may exclude the endpoint, while others will include them.

2. To plot the displacement of the mass, you're going to have to extract the displacements from the list of states returned by the simulation. A list comprehension might be helpful here.

3. The `time` function in the `time` module gives you a timestamp on your computer in seconds. You should think about how you can use this to get the time it takes to run a line of code on your computer. There is no need to use more complicated utilities like `timeit` for this assignment.

4. Your times can be approximate, in the sense that you can have other stuff running on the computer. The goal of this part of the lab is to show you how the times

scale as the number of items being counted in your list grows, not to get the fastest times or show exact timing information. Also, for very small runtimes, Windows systems may show a runtime of 0 because the Windows clock runs slower than the code runs; this is OK.

Grading Checkpoints

Assignment Grading Checkpoints (12 points total)

1. System displacement is plotted [2 point] with a title and axes [1 point].

2. `simulate_gachapon` produces the correct distribution of results [2 points]*. Histogram is plotted with the correct shape [1 point] and a title and axes [1 point].

3. `random_list` works [1 point]*. Code properly records the runtime of `get_element_counts` [1 point] and produces a plot with the right characteristics [1 point] with a title and axes [1 point]. Correct explanation for observed runtime [1 point].

4. **Extra credit** (1 point): Correct explanation/justification for linear/constant time implementation of `get_element_counts`.

Standard Grading Checkpoints (3.5 points total)

- [0.5 points] Turn in a properly filled-in PDF to Gradescope.
- [1 point]* Code passes PEP8 checks with 10 errors max.
- [2 point] Code passes TA-reviewed style checks for cleanliness, layout, readability, etc.

* Autograder will grade this and give you immediate feedback on correctness.
No marker means it will be manually graded.

For this assignment, everything will be turned into Gradescope. **From now on, you will submit your code and PDF together in the same assignment**.

Problem 1: Plot spring damper system

For this assignment, **create two files: `generate_plots.py` and `utils.py`.** Each of the 3 problems will ask you to output plots; all of your plot-generating code should go in `generate_plots.py`, while all functions which are not directly related to plotting should go in `utils.py`.

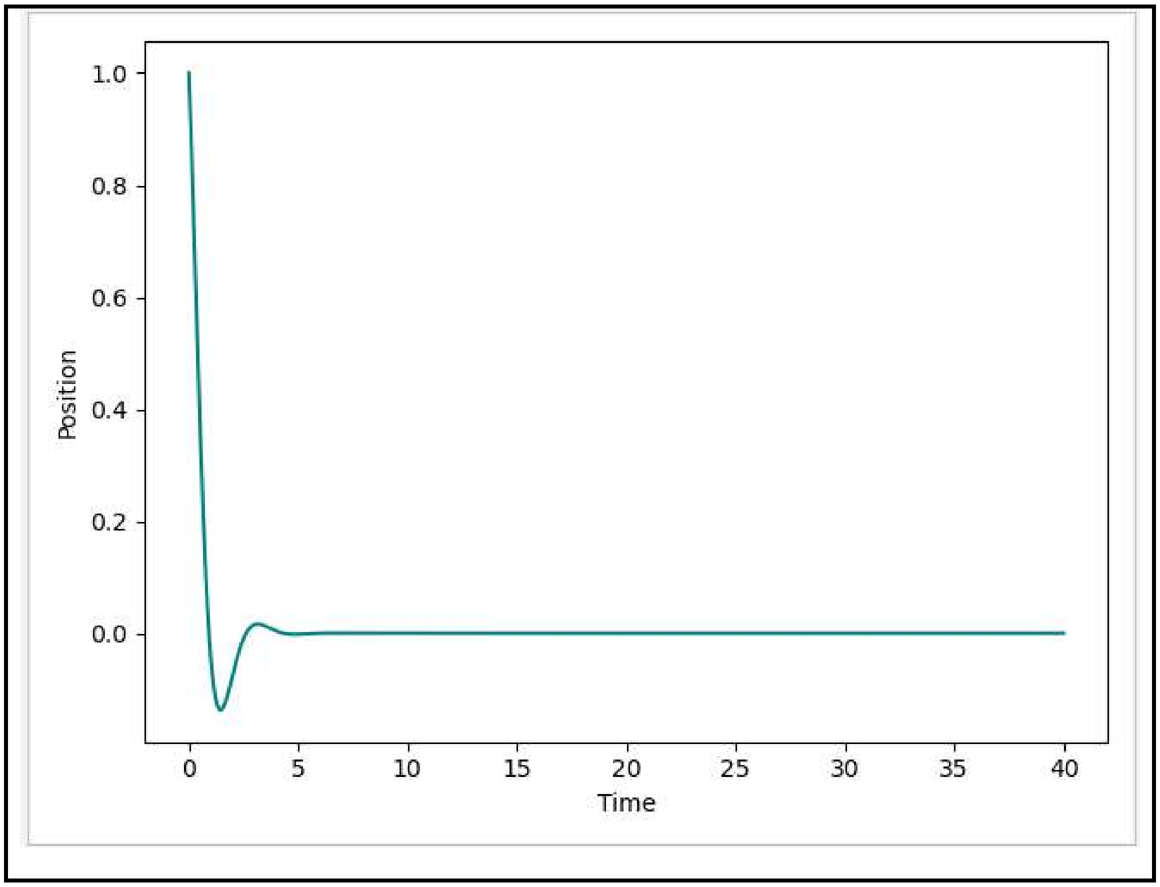
Download the file `msd.py` and take a look at the code inside of it. Do **not** modify this file when doing the assignment.

The `MassSpringDamper` class simulates a mass-spring-damper system, using the ODE integrator that is part of the `scipy` scientific python package. Don't worry if you can't completely understand what's going on right now, since some things, such as classes and lambdas, haven't been covered yet. For now, read the usage information in the docstrings on how to use the system. Then, for this problem, inside of `generate_plots.py`:

- Use the `MassSpringDamper` class to simulate a system with a mass of 1.0, a spring constant (k) of 5.0, and a damper value (c) of 2.5 with a starting position of 1 and a starting velocity of -1. It should go for 40 seconds, using 0.01s time increments.
- Plot the resulting system **displacement** (i.e. position). Ensure that the following specifications are met:
    - The x-axis time values are consistent with the time simulated (i.e. they should be between 0 and 40)
    - Your plot has a title and labelled axes.

Comments for grader/additional information (if any)

Plot output


1.0
0.8
0.6
0.4
0.2
0.0
Position
0
5
10
15
20
25
30
35
40
Time

Problem 2 Histogram: The gachapon problem

This problem considers the following scenario:

> **Your favorite brand of cereal announces a new promotion where each box of cereal you buy contains a random prize from a set of N total prizes. How many boxes of cereal do you have to buy to get them all?**

We will write a function to simulate this, and then plot the resulting distribution. Note that the description here should be sufficient to implement the gachapon problem; you don't need anything beyond a uniform random number generator.

- In `utils.py`, write a function called `simulate_gachapon(n)` which simulates a game with `n` total prizes. Each iteration, randomly generate an integer from 0 to n-1 and add it to your "prize pool". Then, check if you've obtained all the prizes. If not, iterate again. Once you're done, return the number of iterations it took to obtain all the prizes.

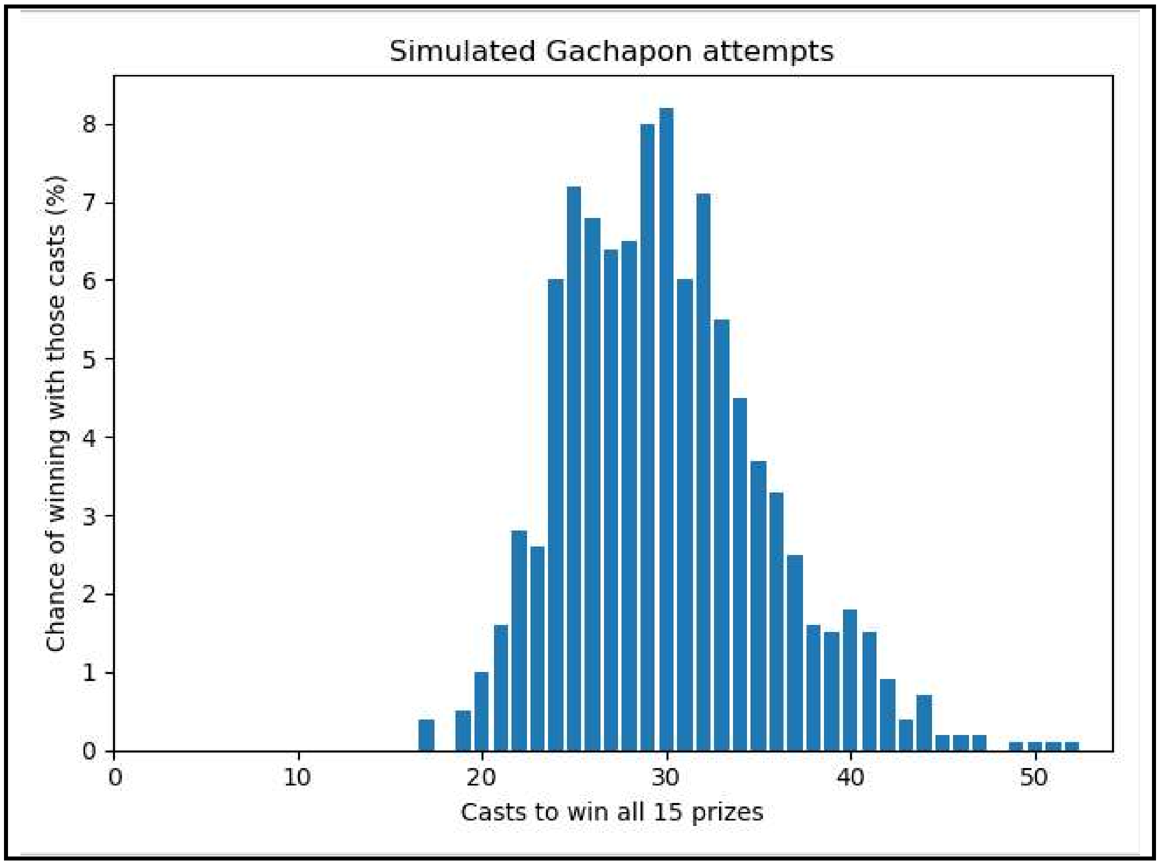
  The `random` module in Python provides many utilities for random number generation. Read Thought 1 in the beginning for some tips.

  (Sanity check: `simulate_gachapon(n)` should never return a number less than `n`.)

- In `generate_plots.py`, write some code which runs `simulate_gachapon(15)` 1000 times. Store down each of the results, and then plot the values in this list as a histogram. Be sure to add a meaningful title and axes. Also be sure the left edge of the plot starts at 0. (Bin size does not matter so long as the shape of the distribution is clear enough.)

Comments for grader/additional information (if any)

Plot output


Simulated Gachapon attempts
Chance of winning with those casts (%)
Casts to win all 15 prizes
0
1
2
3
4
5
6
7
8
0
10
20
30
40
50

Problem 3 Algorithmic Runtimes

For this problem, we're going to take a look at how different code implementations can lead to certain algorithm complexities. Download the provided file `counter.py`, which has several functions in it; the one we care about is `get_element_counts()`. You should look at the function for a bit to try to understand what it's doing.

1. In `utils.py`, write a function called `random_list()` which takes in an integer `n`. It should return a random list of length `n` with integers uniformly sampled from 0 to n-1 (inclusive).

```
# Example:
random_list(10)  # May return [0, 0, 3, 9, 5, 1, 4, 3, 1, 2]
```

2. In `generate_plots.py`, write some code which does the following. For all values 50, 100, …, up to 2500 (call each value n), it should first create a list with `random_list(n)`. It should then time how long it takes for `get_element_counts(your_list)` to run by using the `time()` function in the `time` module. Store the runtimes in a list.

3. Create a **scatter plot** which plots the runtimes on the y-axis against the list lengths on the x-axis. Be sure to have a title and labelled axes.

4. In the provided text box below, comment on the observed trend between runtime and list length. State what you believe the algorithmic complexity is, and point out which part of the code is causing this to be the case.

5. **Extra credit:** In the provided text box below, state if there is an algorithm which can replicate what `get_element_counts` does in *linear time*. If there is, write in words (*not* in Python code) what the algorithm is, and justify why it's linear time. If there is not, justify why there isn't one. Then, do the same analysis for *constant time*.

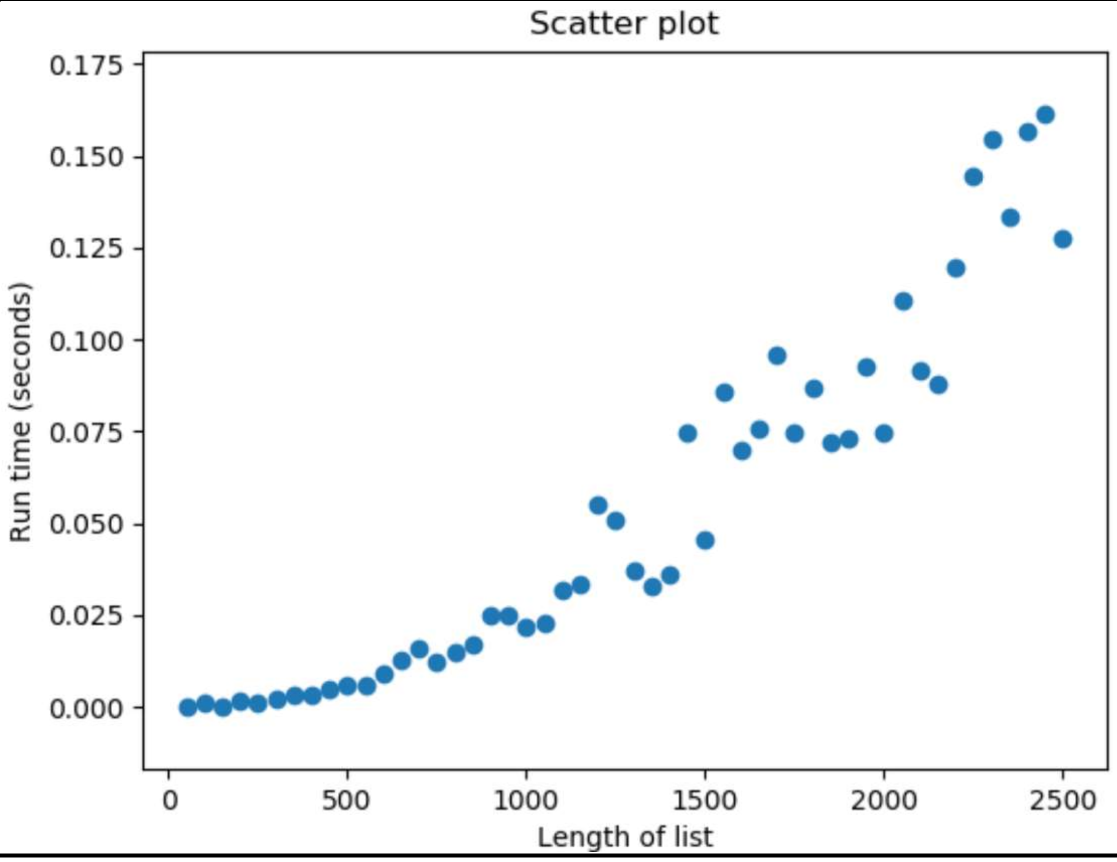
Comments for grader/additional information (if any)

Part 4: Running time
What is the running time in terms of the number of elements? Which part of the code causes this? Run time appears to be near n^2. If I had to guess the part of the code that causes this is the for loop in get_element_counts.

Part 5 (Extra credit): Improvements

```
Describe linear/constant-time implemenations for get_element_counts
(in words, not with Python code) or justify why they don’t exist.
```

Plot


Scatter plot
Run time (seconds)
Length of list