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

Names of people you worked with:
•

Websites you used:
• https://stackoverflow.com/questions/13479163/round-float-to-x-decimals/22155830 • https://www.pythonforbeginners.com/system/python-sys-argv • https://docs.python.org/3/library/sys.html • https://realpython.com/python-dicts/ • https://www2.clarku.edu/faculty/djoyce/complex/mult.html

Approximately how many hours did it take you to complete this assignment (to nearest whole number)?	9

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.

Turn .zip files of Python code to Gradescope or your assignment will not be graded

BEFORE YOU BEGIN

1. Create two files, complex.py and gachapon.py
2. Read up on complex numbers if you've forgotten how they work
3. Read chapters 15 and 16 in the text book

Learning Objectives:

You should be able to do the following:

- Make a class
- Add class attributes and have them store immutable and mutable data
- Override the base members such as repr, +, str by using magic methods
    - See notes from lab

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

1. Python already has support for complex numbers. We're going to ignore that for the purposes of this assignment. **Your code should not use any built-in Python functionality for complex numbers** (except possibly for testing).

2. `__repr__` and `__str__` do similar things. Python internally uses `__repr__` sometimes and `__str__` at other times. You should implement `__repr__` to return the string representation of the complex number class. Then have `__str__` have a single line: `return self.__repr__()`

Grading Checkpoints

Assignment Grading Checkpoints (12 points total + 2 extra credit)

1. GachaponSimulator initializes correctly [1 point]*.
2. `_simulate_once` works and produces the expected number [0.5 points]*.
3. `reset` works. [0.5 points]*
4. `simulate` works and extends the `results` list with the results. [1 point]*
5. `get_summary_stats` work as specified (1 point)*.
6. Command line interface which allows user to pass in number of prizes and iterations works (1 point).
7. Complex initialization works, with values saved correctly. [2 point]*
8. `__repr__` and `__str__` work as described [1 points total] *
9. Addition works and produces the right value for two Complex objects (1 point), a Complex plus a number (0.5 points), and a number plus a Complex (0.5 points) [2 points total] **
10. Multiplication works for all three cases. [2 points] **
11. **Extra credit:** Subtraction (0.5 points) and division (1.5 points) work for all three cases. **[2 extra credit points]****

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 inform you of correctness.
** Autograder will check to make sure that the operators are functional, but will not tell you if the number returned is right.
^ Autograder will grade but hide output until after grades are released.
Blank indicates the problem will be manually graded.

For the style checks, we are looking for the following:

- Proper structuring of the class, e.g. not writing to global variables, proper encapsulation, etc.
- Proper utilization of data structures, e.g. using my_obj.my_method() instead of MyClass.my_method(my_obj)
- General code cleanliness, e.g. variable naming, comments, no extraneous code, etc.

Turn in your code and this PDF to Gradescope.

Problem 1 Gachapon Simulator Revisited

Recall from Lab 4 we wrote a function `simulate_gachapon` that simulated the number of iterations it took to win all N prizes out of a selection of N prizes, assuming each prize was drawn uniformly with replacement. The objective here is to take this code and package it into a compact class which is easy to interact with. Create a new file `gachapon.py` for this assignment.

1. Write a class `GachaponSimulator` which initializes with 1 parameter `prizes_n`, representing the size of the prize pool. You should save it in an appropriately named attribute. You should also create an attribute called `results` which defaults to an empty list.

   Note: You are not yet running the simulator when you initialize something like `our_sim = GachaponSimulator(10)`; you are simply "laying the groundwork", so to speak.

2. Write a method called `_simulate_once` which takes in no additional arguments (besides `self`) that runs one round of the gachapon game. It should return the number of iterations the game took.

3. Write a method called `reset` which takes in no additional arguments which resets the `results` attribute to an empty list.

4. Write a method called `simulate` which takes in one additional argument `num_games` representing the number of games to simulate. It should then run that number of games and record the number of iterations for each game. **It should then take these results and append them to the object's `results` list.**

   Note that the `results` list should NOT be reset if `simulate` is called multiple times. This means that if we call `our_sim.simulate(30)` and then `our_sim.simulate(40)`, then `len(our_sim.results)` should be 70.

   Also note that this method is not required to return anything.

5. Write a method called `get_summary_stats` which takes in no additional arguments. It should return a dictionary with the number of games which have currently been run (with key `n`), the mean number of iterations across all the games (with key `mean`), and the population standard deviation (with key `stdev`) from the current results list.
   a. Key is the .simulate(n) ← tells you how many times it ran the simulation
   b. Key mean gets the mean value
   c. Key stdev

      If the user has not run any games, both `mean` and `stdev` should be the

value `None`. If the user has only run one game, `stdev` should be `None`.

(Hint: numpy has a mean and standard deviation function you may wish to use.)

6. Finally, implement a command line interface at the end of `gachapon.py` which lets you pass in two arguments, a number of prizes and a number of games to play, and that will print out a nicely formatted message telling the user the results of the simulation. The command would look like:

   ```
   python gachapon.py 10 1000
   ```

   A possible output would be:

   ```
   Running 10-prize lottery simulator 1000 times…
   Average number of iterations was 32.15 (standard deviation of
   5.18)
   ```

   You should take advantage of the interfaces defined in `GachaponSimulator` to reduce the amount of code for this section; the process of actually running the simulation and obtaining the stats should be just a few lines of code.

Comments for grader/additional information (if any)

Problem 2 Complex class - Setup

In this section, we'll write a class which represents a complex number. Note that Python already has support for complex numbers built in; for this example, we're expecting you to reimplement the basic functionality from scratch. All work should go in a file called `complex.py`.

1. Create a `Complex` class that initializes with two *optional* parameters representing the real and imaginary components, defaulting to 0 if they are not specified. Store the real and imaginary components as floats in attributes called `re` and `im`. Therefore, we should be able to do something like:

```
a = Complex(-3, 2)
b = Complex(2)
print(a.re)  # Should print -3.0
print(b.im)  # Should print 0.0
```

2. Implement the `__repr__` and `__str__` magic methods so that when we try to print out your Complex number, we get a readable representation of your complex number, exactly following the format shown below:

```
In[1]:  print(Complex(-3, 2))
Out[1]: (-3.0 + 2.0i)

In[2]:  print(Complex())
Out[2]: (0.0 + 0.0i)

In[3]:  print(Complex(3.4,-2.1))
Out[3]: (3.4 - 2.1i)
```

Comments for grader/additional information (if any)

Problem 3 Complex class - Arithmetic

In this section, we will extend your `Complex` class to be able to do basic arithmetic. You should be able to add Complex objects with other Complex objects, but you should also be able to add them to other integers and floats.

While we will not grade you on test code for this section, we highly encourage you to write some test code to make sure everything is operating as expected. **The autograder will not show you correctness results until after the assignment is graded.**

1. Start by implementing addition, so that all of the following work:

   ```
   a = Complex(2.0, 3.0)

   print(a + Complex(-1.5, 2))  # (0.5 + 5.0i)
   print(a + 8)                 # (10.0 + 3.0i)
   print(3.5 + a)               # (5.5 + 3.0i)
   ```

   Note that the result of the addition operation should be another Complex object. You will need to define the `__add__` and `__radd__` magic methods to get these to work.

2. Next, implement multiplication, so that all of the following work:

   ```
   a = Complex(1.0, -3.0)

   print(a * Complex(4.0, 5.5))  # (20.5 - 6.5i)
   print(a * 3.5)                # (3.5 - 10.5i)
   print(-2 * a)                 # (-2.0 + 6.0i)
   ```

3. **Extra Credit:** Implement subtraction and division in the same vein. Note that because subtraction and division are non-commutative, you should pay special attention to how you define the reverse magic methods. (Did we mention to test your code?)

   Note: The magic method for division is `__truediv__`, for legacy reasons.

Comments for grader/additional information (if any)