CMPSC-132: Programming and Computation II
Spring 2020

Lab #2

Due Date: 02/07/2020, 11:59PM EST

Read the instructions carefully before starting the assignment. Make sure your code follows the stated guidelines to ensure full credit for your work.

Instructions:

- The work in this lab must be completed alone and must be your own.
- **Watch the video lectures before you start this assignment**
- Download the starter code file from the LAB2 Assignment on Canvas. **Do not change the function names or given started code on your script.**
- A doctest is provided as an example of code functionality. Getting the same result as the doctest does not guarantee full credit. You are responsible for debugging and testing your code with enough data, you can share ideas and testing code during your recitation class. As a reminder, Gradescope should not be used to debug and test code!
- Each function must return the output (Do not use print in your final submission, otherwise your submissions will receive a -1 point deduction)
- **Do not include test code outside any function in the upload. Printing unwanted or ill-formatted data to output will cause the test cases to fail. Remove all your testing code before uploading your file (You can also remove the doctest). Do not include the input() function in your submission.**

Goal:

[5 pts] Write the class *VendingMachine* that will represent a typical vending machine (item number, price, stock). An instance of *VendingMachine* has access to five methods, *newStock*, *getStock*, *purchase*, *deposit* and *restock* that will describe its interaction with the user **returning** strings. This vending machine does not give you money back if you don't make a purchase. *Tip*: use the string *format* method to construct the strings

Vending Machine notes:

- When creating an instance of this class, the machine is empty, thus, calling *getStock, purchase and deposit* simply return out of stock messages:

```
>>> x=VendingMachine()
>>> x.purchase(215)
'Machine out of stock'
>>> x.deposit(10)
'Machine out of stock. Take your $10 back'
>>> x.getStock()
'Machine out of stock'
>>> x.restock(215, 9)
'Invalid item'
```

- *newStock* is used only when the registering a new item. If an item is out of stock or still has stock, using *newStock* will return a message to the user. For already registered items, you use the *restock* method. *Restock* returns an error when the item is not in the dictionary

```
>>> x.newStock(215,2.50,2)
'Current 215 stock: 2'
```

```
>>> x.restock(215, 1)
'Current item stock: 3'
>>> x.restock(5, 1)
'Invalid item'
>>> x.getStock()
{215: [2.5, 3]}


# User deposits 7.5 and purchases 3 items of 215

>>> x.getStock()
{215: [2.5, 0]}
>>> x.newStock(215,2.50,2)
'Item already registered'
>>> x.newStock(156,3,3)
'Current 156 stock: 3'
>>> x.getStock()
{215: [2.5, 0], 156: [3, 3]}
```

- If all products are out of stock, deposit returns an out of stock message

```
>>> x.getStock()
{215: [2.5, 0], 156: [3, 0]}
>>> x.deposit(10)
'Machine out of stock. Take your $10 back'
```

- *purchase* uses quantity=1 as a default value, which can be overwritten by providing the second argument. Before completing a purchase, machine has to check if the item is valid, if it is on stock and if the user has enough balance.

```
>>> x.getStock()
{215: [3, 1], 156: [3.5, 4]}
>>> x.purchase(56)
'Invalid item'
>>> x.purchase(215)
'Please deposit $3'
>>> x.deposit(1)
>>> x.purchase(215)
'Please deposit $2'
>>> x.deposit(10)
>>> x.purchase(215)
'Item dispensed, take your $5'
>>> x.purchase(215)
'Item out of stock'
>>> x.getStock()
{215: [3, 0], 156: [3.5, 4]}
```

- *getStock* returns a dictionary where the key is the item and the value is the list [price, stock]
- *purchase* will ask for the needed money to complete a purchase if enough items are in stock, otherwise, it returns the current stock of the item (for example 3 items in stock and attempting to purchase 4)
- Once the user completes one purchase, machine returns remaining balance
- For more examples of functionality, check the doctest in the starter code
- Quotes mean method returned a string, no need to append them in your code
- Return output with the sentences provided. Solution is not case sensitive, which means Balance: $10 is the same as balance: $10, but is not the same as Balance= $10

[5 pts] Write the class *Complex* that supports the basic complex number operations. Such operations are addition (+), subtraction (-), multiplication (*) and division (/) of complex numbers, and addition (+), subtraction (-) and multiplication (*) of a complex by a scalar (float or int). All methods must **return** (not print) the result. Class also supports the rich comparison for equality (= =)

- Check the doctest for object behavior examples.
- You must use the special methods for those 5 operators in order to override their behavior.
- You will need other special methods to achieve a **legible object** representation (what you see when printing and returning the object).
- Addition, subtraction and multiplication by scalar and by another complex number use the same operator, so you must check the type of the object in order to decide which operation you have to perform
- **Rich comparison returns a Boolean value**
- **The rest of the methods must return a new Complex object**, this means the original objects should not change after the operation is performed.
- Test your code, this is how you ensure you get the most credit out of your work!!
- When performing an invalid operation, return None
- **You are not allowed to modify the constructor or any given code.**
- Note that to support *number* (+, - or *) *Complex*, you are going to need 3 other special methods that can reuse code from the *Complex* (+, - or *) *number* operations. **Hint**: Section 3.3.1. Basic customization (__*eq*__) and Full Section 3.3.8. Emulating numeric types in https://docs.python.org/3/reference/datamodel.html#emulating-numeric-types

In addition, write the property method conjugate into the Complex class. The complex conjugate of the complex number $z = x + y$i is given by $x - y$i

Remember, **you are not allowed to modify the given constructor**. The method *conjugate* **must be a property method, otherwise, no credit will be given**. Property methods are discussed in the video lectures!

You can visit https://www.varsitytutors.com/hotmath/hotmath_help/topics/operations-with-complex-numbers for complex number formulas

General note: Start by watching the lectures. Do not overthink this assignment, you don't have to come up with a complex algorithm to solve it. Just follow the formulas and directions, the purpose of this assignment is to familiarize you with object-oriented programming syntax

Deliverables:

- Submit your code with the file name LAB2.py to the Lab2 Gradescope assignment before the due date