Homework 2

- Due 10/14 23:59 pm est
- Following instructions provided in **Homework submission instructions**
- Do not use any external modules on this assignment

Problem 1 (30pts)

For this problem, you will create a **Point** class that can be used to generated any point in the cartesian coordinate. The followings are the attributes and methods that should be implemented in the **Point**:

- `__init__` (already done): To initialize a point object, you need to specify the x and y coordinates and assign to `self.x` and `self.y` attributes.
- Comparators methods:
  - Support the comparators < (`__lt__`), >(`__gt__`), and ==(`__eq__`).
  - p1 is less than p2 if its distance from the origin is less.
  - p1 is greater than p2 if its distance from the origin is greater.
  - p1 and p2 are equal if they are the same distance from the origin.
- `dist_from_origin` method:
  - Returns the cartesian distance of this point from the origin

In [ ]:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Problem 2 (50pts)

We have seen how to construct the **CheckingAccount** in the class. In this problem, you will construct the **SavingAccount**. Similar to the CheckingAccount class, since SavingAccount **is a** Account, so it should have all the attributes and methods defined for Account class. In addition, it should also have it's unique attributes and methods:

- The interest_rate for saving account is 10%.
- The minimum for saving account is 1000.
- Since rewards of a savings account were greater than those of a checking account, the bank allowed only one withdrawal from a savings account.
- When customer close the account, to reward the savings account owners, the bank added additional bonus of 15% (calculated off of the minimum amount held throughout the year) PLUS a fixed amount of 100 . Hence, for example, if the holdings were, 1000,

800, 1200, the owner of a savings account got 1200 + 800 x 0.1 + 800 x 0.15 + 100 = 1500.

Requirements:

Your SavingAccount class should satisfy the followings:

- `__init__`:
    - To initialize a SavingAccount object, we need to specify (Already done):
        - initial_amount
        - max_num_withdrawals (default: 1)
        - minimum (default: 1000)
        - interest_rate (default: 0.1)
        - bonus_contribution (default: 0.15)
    - Inherit all the attributes and methods from its super class Account (Already done).
    - Additional attributes that are unique to SavingAccount:
        - self._num_withdrawals (set to 0 when initializing)
        - self._max_num_withdrawals
        - self._bonus_contribution
- `get_num_withdrawals` method:
    - Return the number of withdrawals.
- `withdraw` method:
    - Override the withdraw method from its super class.
- `add_bonus` method:
    - Increase the amount held by (percent bonus contribution) * (minimal amount ever held) + 100.
- `close_account` method:
    - Override the close_account method from its super class.

Remark

- import uuid module using `import uuid`.
- import Account class from the module named "Account" using `from Account import Account` (Make sure to put the Account.py file in the same folder as your homeowork file).
- Use the chunk below to design you **SavingAccount** class. The comments are served as instructions for each methods. Use those instructions and insert you code right below each comment.

In [ ]:

```python
import uuid
from Account import Account
class SavingAccount(Account):
    def __init__(self, initial_amount, max_num_withdrawals=1,
                 minimum=1000, interest_rate=0.10,
                 bonus_contribution=0.15):
```

```
        super().__init__(initial_amount, minimum, interest_rate)

        # 1) Set the number of withdrawals to 0 (only SavingsAccounts track the
        #    of withdrawals

        # 2) Set the _max_num_withdrawals attribute to the value given in the a
        #    of the constructor

        # 3) Set the _bonus_contribution to the value given in the argument of
        #    the constructor


    def get_num_withdrawals(self):
        # Simply return the number of withdrawals


    def withdraw(self, w_amount):
        # 1) If the number of withdrawals is >= than the maximal number of
        #    withdrawals allowed throw and exception:
        #    raise ValueError("Savings accounts allow only {} withdrawals.".for

        # 2) Increase the withdrawal counter by 1

        # 3) Call the parent's implementation of withdraw as it does the rest c
        #    things for us
        #


    def add_bonus(self):
        # According to the banks rewards scheme, increase the amount held by th
        # (percent bonus contribution) * (minimal amount ever held) + 100


    def close_account(self):
        # 1) Add bonus

        # 2) Call the parent's close_account method as it does lots of stuff al
        #    return super().close_account()
```

Problem 3 (20 pts)

For All the methods you defined in above, write a test to test they do what you expect them to do. For example, to test the `__lt__` method defined in Point class, we can do the following.

In [ ]:

```
import unittest
class TestMyCode(unittest.TestCase):
    def testComparatorlt(self):
        point1 = Point(3, 4)
        point2 = Point(1, 2)
        self.assertEqual(point1<point2, False)
```

Requirements

- Define you own test inside the above class to test othr methods you have written in Problem 1 and 2.
- Create one test for each method.