CSCI 1133, Fall 2019
Programming Assignment 10
Due: 11:55pm, Wednesday November 13, 2019

Due Date: Submit your solutions to GitHub by 11:55 p.m., Wednesday, November 13th. We will do a pull from this time point. Do not upload anything to Canvas and PLEASE be sure to use proper naming conventions for the file, classes, and functions. We will NOT change anything to run it using our scripts.

Unlike the computer lab exercises, this is not a collaborative assignment. You must design, implement, and test your code on your own without the assistance of anyone other than the course instructor or TAs. In addition, you may not include solutions or portions of solutions obtained from any source other than those provided in class (so you are ONLY allowed to reuse examples from the textbook, lectures, or code you and your partner write to solve lab problems). Otherwise obtaining or providing solutions to any homework problem for this class is considered Academic Misconduct. See the syllabus and read section "Academic Dishonesty" for information concerning cheating. Always feel free to ask the instructor or the TAs if you are unsure of something. They will be more than glad to answer any questions that you have. We want you to be successful and learn so give us the chance to help you.

Instructions: This assignment consists of 2 problems, worth a total of 40 points. Solve the problems below by yourself, and put all functions in a single file called `hw10.py`. Use the signatures given for each class and function. We will be calling your functions with our test cases so you must use the information provided. If you have questions, ask!

Because homework files are submitted and tested electronically, the following are very important:

- You follow all naming conventions mentioned in this homework description.
- You submit the correct file, hw10.py, through Github by the due date deadline.
- You follow the example input and output formats shown.
- Regardless of how or where you develop your solutions, your programs should execute using the python3 command on CSELabs computers running the Linux operating system.

Push your work into Github under your own repo. The specific hosting directory should be: repo-<username>/hw10, where you replace <username> with your U of M user name. For instance, if your email address is bondx007@umn.edu, you should push your hw10 to this directory: repo-bondx007/hw10

The following will result in a score reduction equal to a percentage of the total possible points:

- Incorrectly named/submitted source file, functions, or classes (20%)
- Constraints not followed (40%)
- Failure to execute due to syntax errors (30%)

Use the following template for EVERY function **or method** that you write that requires at least 5 lines of code. The template should appear just before the function/method.

```
#=========================================
# Purpose: (What does the function do?)
# Input Parameter(s): (Each parameter by name and what it represents)
# Return Value(s): (What gets returned? Possibilities?)
#=========================================
```

Problem A. (10 *points*) **Complex Class**

In this problem you need to create a class named Complex that models a complex number, numbers of the form x+yi, where x and y are real numbers, and i is the imaginary unit equal to the square root of -1. In the x+yi representation, we say that x is the real component of the complex number, and y is the imaginary component. Note that Python already has a built-in Complex class, but this makes for a good, simple example, so we're going to make our own. Obviously, you're not allowed to use the built-in Complex class to implement your Complex class.

Complex objects have two instance variables:

- `real` (a numeric value representing the real component of the complex number)
- `imag` (a numeric value representing the imaginary component of the complex number)

(Note: **Please do not write all of these methods at once.** Copy the test cases in the section below this into your hw10.py file, and uncomment a few of them at a time as you implement things: make sure that each method works before moving on to the next one. This is something you should be doing anyway but is especially important when writing long pieces of code like class definitions)

The Complex class must include the following:

- `__init__(self, real, imag)`: A constructor with two numeric arguments (not counting self), that initialize the real and imag instance variables, respectively.

- You must have getter and setter methods for both `real` and `imag`. They should have the following signatures:
  - `get_real(self)`
  - `get_imag(self)`
  - `set_real(self, new_real)`
  - `set_imag(self, new_imag)`

- Overload the `__str__(self)` method so that it returns a string of the format:

  "*<real>* + *<imag>*i"

  where *<real>* and *<imag>* represent the real and imaginary components of the complex number, respectively.

- Overload the + and * operators, so that they take in a Complex object `other`, and return a new Complex object representing the sum or product (respectively) of the complex numbers represented by `self` and `other`. See the Hints section if you're not familiar with how to add or

multiply two complex numbers. You'll need to have the following method signatures to overload the operators:
    - `__add__(self, other)`
    - `__mul__(self, other)`

- Overload the == operator, so that it takes in a Complex object `other`, and returns `True` if the two objects are equal (their real components are equal, and their imaginary components are equal), or `False` otherwise. You'll need to have the following method signature to overload the operator:
    - `__eq__(self, other)`

Hints:

- Review on how to add/multiply complex numbers:
    - (a + bi) + (c + di) = (a + c) + (b + d)i
    - (a + bi)(c + di) = (ac - bd) + (ad + bc)i

Constraints:

- Do not import/use any library modules.
- Your submission should have no code outside of class/function definitions (comments are fine).
- You are not permitted to use Python's built-in Complex class
- Your methods must have exactly the same names and arguments as described above.

Example: Copy the following code into your hw10.py file, and uncomment lines as you write the relevant methods. What each line should print is noted in the comment on the right. You should write some additional tests of your own: this is not a very good set of tests for the number of methods this problem requires. Make sure to comment out all tests before submitting to github.

```
##x = Complex(2.7,3)
##y = Complex(-4,0)
##z = Complex(0,-1.5)

##print(x.real)             # 2.7
##print(x.get_real())       # 2.7
##print(y.get_imag())       # 0

##x.set_real(6)
##z.set_imag(-2.5)

##print(x.real)             # 6
##print(x)                  # 6 + 3i
##print(y)                  # -4 + 0i
##print(z)                  # 0 + -2.5i
##print(x+y)                # 2 + 3i
```

```
##print(z+y+x)              # 2 + 0.5i
##print(y*z)                # 0.0 + 10.0i
##print(x*z)                # 7.5 + -15.0i
##print(x*y+x*z)            # -16.5 + -27.0i
##print(x*(y+z))            # -16.5 + -27.0i
##print(x == y)             # False
##print(x*y+x*z == x*(y+z))   # True
```

Problem B. (30 *points*) **Employee Database**

You work for the HR department, in the corporate headquarters of Holistic Synergies, Ltd. Your supervisor, Dr. Boss Manager III, has instructed you to "leverage our operational efficiency to create unique opportunities for strategic alliances, so that the full human resources portfolio, from data to incentives to programs and beyond, can be the most innovative across the industry, but in ways that also reflect our leading stewardship of resources and cost-effective sensibility."

You have no idea what this means, so instead you've decided to make a program that processes information about the various branches of your company. Your company cuts employees with alarming frequency, so you've also decided to automate the decision-making process for those tasks with your program (note: this is sort of automation is ethically dubious at best, please don't do this for any real company).

Download the CSV files contained in `hw10.zip` on Canvas, and place them in the same folder as your `hw10.py` file. Each CSV file represents one branch of the company. Note the layout of the files: the first two rows contain information about the location of the branch and the annual upkeep cost for the building (in USD) respectively. The third row is a set of column titles for the employees, and the remaining rows contain information about each employee.

You may notice that every employee is mentioned only by first name; it is against company policy to have a last name, as this "disincentivizes exceptional distillation of latent mnemonic energies into innovative crystallization", according to your CEO. For similar reasons, it is against company policy to hire more than one person with the same first name, even across different branches.

You will need three interconnected classes for this system: one for a single employee, one for a branch, and one for the entire company. Since the information for each branch is in a file, and each line of that file (past the 3rd) is an employee, you'll be using that information directly in your initialization of the employee and branch classes.

`Employee` class

The Employee class must have five instance variables:

- `name` is a string representing the employee's first name (corresponding to the first column in the CSV, from row 5 onwards)
- `position` is a string representing the employee's job title (the second column)
- `salary` is a floating point number representing the employee's annual salary in USD (the third column). This must be stored as a float value, not a string (use the float function), and the same applies to the next two as well.
- `seniority` is a floating point number representing the number of years the employee has worked at that branch of the company (the fourth column).
- `value` is a floating point number representing an estimate of the average annual earnings that the employee generates for the company (not taking into account his salary); that is, how much the employee is worth to the company (the fifth column)

The Employee class should also have a constructor and three methods:

- The constructor must take only one parameter (other than self), a single string representing the line in the CSV file that contains the employee's data. The constructor should parse that line using the `.split` string method to initialize all five instance variables mentioned above.
- Overload the `__str__` method so that it returns a string in the format
  `"<Name>, <Position>"`
  (see example below)
- `net_value(self)` takes in no arguments (other than self), and returns the employee's value, minus their salary.
- Overload the < operator (`__lt__`) so that it takes in another `Employee` object, and returns `True` if the left operand (self) has a lower net_value than the right operand (other), or `False` otherwise.

Examples: Copy the following code into your hw10.py file, and uncomment lines as you write the relevant methods. What each line should print is noted in the comment on the right. Make sure to comment out all tests before submitting to github. You may run into minor floating point rounding errors - ignore them.

```
##emp1 = Employee('Milton,Underling,310423.01,5.0,22.99\n')
##emp2 = Employee('Bill,Boss,403567.34,5.0,519.35\n')

##print(emp1.name)             # Milton
##print(emp1.position)         # Underling
##print(emp1.salary)           # 310423.01
##print(emp1.seniority)        # 5.0
##print(emp1.value)            # 22.99
##print(emp1)                  # Milton, Underling
##print(emp1.net_value())      # -310400.02
##print(emp2.net_value())      # -403047.99
##print(emp1 < emp2)           # False
##print(emp2 < emp1)           # True
```

`Branch` class

The Branch class must have three instance variables:

- `location` is a string, representing the city in which the branch is located
- `upkeep` is a float, representing the annual upkeep cost of the building
- `team` is a list of Employee objects, representing every employee working at the branch.

The Branch class should also have a constructor and four methods:

- The constructor must take only one parameter (other than self), a string representing the filename for the CSV where the branch information is stored. The constructor should then open that file, read in each line, and use that information to initialize all of the instance variables for the Branch.
- Overload the `__str__` method so that it returns a string containing the location of the branch, followed by the string representation of each employee (order does not matter), separated by

newlines (see example below). You must call the str() function on each Employee object to do this.

- `profit(self)` takes in no arguments (other than self), and returns the sum of the net values of all of the employees in the branch, minus the upkeep of the branch.
- Overload the < operator (`__lt__`) so that it takes in another `Branch` object, and returns `True` if the left operand (self) has a lower profit than the right operand (other), or `False` otherwise.
- `cut(self, num)` takes in one argument: `num` is an integer representing the number of employees that must be cut from the branch. `cut` sorts the employees by their net value, and removes the lowest `num` Employees from the team list (see Hint below for how to do this quickly).

Hint:

- There are two built-in ways of quickly sorting lists: the `sorted()` function, and the `.sort()` list method; see https://docs.python.org/3/howto/sorting.html. The difference is that `.sort()` is in-place: it doesn't return anything but will change the list to be in sorted order, while `sorted()` does not change the original list: `sorted()` creates a copy, sorts the copy, and returns it. By default, `.sort()` and `sorted()` sort objects based on their overloaded < operator (`__lt__`), so because we changed the behavior of < to order Employees based on their net value, you should be able to use `.sort()` or `sorted()` on a list of Employees without specifying a key function.

Examples (assumes that you are running hw10.py from the same folder as all of the CSV files in hw10.zip, which can be found on Canvas)**:** Copy the following code into your hw10.py file, and uncomment lines as you write the relevant methods. What each line should print is noted in the comment on the right, or to the right and below when printing entire branches (don't uncomment the lines starting with #### since those are supposed to remain comments). Make sure to comment out all tests before submitting to github. You may run into minor floating point rounding errors - ignore them. Remember, the order in which the employees are listed does not matter when printing out a Branch object.

```
##branch2 = Branch('branch2.csv')
##print(branch2.location)          #Pawnee
##print(branch2.upkeep)            #98229.98
##print()
##
##print(branch2)
####                Pawnee
####                Ron, Efficiency Logistics Strategist
####                Leslie, Creative Evolution Strategist
####                Ann, Data Evolution Engineer
####                Mark, Operational Services Specialist
####                Tom, Creative Logistics Coordinator
####                April, Enterprise Services Consultant
####                Andy, Creative Logistics Specialist
####                Jerry, Enterprise Services Technician
```

```
####                Donna, Operational Innovation Engineer
##
##print()
##branch2.cut(7)
##print(branch2)
####                Pawnee
####                Leslie, Creative Evolution Strategist
####                Ann, Data Evolution Engineer
##
##print()
##print(branch2.profit())         #12577.81
##branch1 = Branch('branch1.csv')
##print(branch1.upkeep)           #57867.07
##
##print()
##print(branch1)
####                Scranton
####                Dwight, Efficiency Evolution Specialist
####                Jim, Data Innovation Specialist
####                Pam, Operational Analytics Technician
####                Ryan, Data Evolution Consultant
####                Stanley, Efficiency Logistics Technician
####                Michael, Enterprise Communications Technician
####                Kevin, Operational Services Technician
####                Meredith, Data Evolution Technician
####                Angela, Enterprise Communications Consultant
####                Oscar, Creative Innovation Coordinator
####                Phyllis, Enterprise Analytics Technician
##branch1.cut(8)
##
##print()
##print(branch1)
####                Scranton
####                Pam, Operational Analytics Technician
####                Michael, Enterprise Communications Technician
####                Stanley, Efficiency Logistics Technician
```

`Company` class

The Company class must have two instance variables:

- `name` is a string, representing the name of the company

- `branches` is a list of `Branch` objects, representing all of the branches associated with the company

The Company class should also have a constructor and two methods:

- The constructor must take two parameters (other than self): a string representing the name of the company, and a list of Branch objects, representing the branches of the company: it should use this information to initialize the two instance variables.
- Overload the `__str__` method so that it returns the company name, followed by the string representation of each branch (in no particular order), separated by two newlines (so one empty line between each branch). See the example below for details.
- `synergize(self)` takes in no arguments (other than self), and implements the company's ultimate HR strategy: find the branch with the lowest profit margin and cut half (rounded down) of the employees from that branch.

Constraints:

- Do not import/use any Python modules.
- Don't use the input() function, as this will break our grading scripts.
- You may use any string or list method that is appropriate to solving this problem.
- Your submission should have no code outside of the function definitions (comments are fine).

Examples: Copy the following code into your hw10.py file, and uncomment lines as you write the relevant methods. What each line should print is noted in the comment on the right, or to the right and below when printing entire branches (don't uncomment the lines starting with #### since those are supposed to remain comments). Make sure to comment out all tests before submitting to github. You may run into minor floating point rounding errors - ignore them. Remember, the order in which the branches are listed does not matter when printing out a Company object.

```
##b1 = Branch('branch1.csv')
##b2 = Branch('branch2.csv')
##b3 = Branch('branch3.csv')
##b4 = Branch('branch4.csv')
##hs = Company('Synergistic Management Solutions',[b1,b2,b3,b4])
##print(hs.name)           #Synergistic Management Solutions

##print()
##print(hs)
####                    Synergistic Management Solutions
####
####                    Scranton
####                    Dwight, Efficiency Evolution Specialist
####                    Jim, Data Innovation Specialist
####                    Pam, Operational Analytics Technician
####                    Ryan, Data Evolution Consultant
####                    Stanley, Efficiency Logistics Technician
####                    Michael, Enterprise Communications Technician
```

```
####                    Kevin, Operational Services Technician
####                    Meredith, Data Evolution Technician
####                    Angela, Enterprise Communications Consultant
####                    Oscar, Creative Innovation Coordinator
####                    Phyllis, Enterprise Analytics Technician
####
####                    Pawnee
####                    Ron, Efficiency Logistics Strategist
####                    Leslie, Creative Evolution Strategist
####                    Ann, Data Evolution Engineer
####                    Mark, Operational Services Specialist
####                    Tom, Creative Logistics Coordinator
####                    April, Enterprise Services Consultant
####                    Andy, Creative Logistics Specialist
####                    Jerry, Enterprise Services Technician
####                    Donna, Operational Innovation Engineer
####
####                    Greendale
####                    Britta, Enterprise Services Coordinator
####                    Abed, Operational Services Strategist
####                    Troy, Efficiency Innovation Coordinator
####                    Annie, Efficiency Innovation Consultant
####                    Shirley, Enterprise Innovation Engineer
####                    Pierce, Enterprise Evolution Specialist
####                    Ben, Enterprise Communications Consultant
####                    Jeff, Brand Logistics Specialist
####                    Craig, Enterprise Communications Engineer
####
####                    Ambiguous
####                    J.D., Efficiency Analytics Coordinator
####                    Turk, Efficiency Innovation Technician
####                    Elliot, Data Innovation Specialist
####                    Perry, Efficiency Evolution Technician
####                    Bob, Enterprise Logistics Engineer
####                    Carla, Brand Services Technician
####                    Janitor, Efficiency Evolution Coordinator

##print()
##hs.synergize()
##print()

##print(hs)
####                    Synergistic Management Solutions
####
####                    Greendale
```

```
####                    Ben, Enterprise Communications Consultant
####                    Craig, Enterprise Communications Engineer
####                    Britta, Enterprise Services Coordinator
####                    Pierce, Enterprise Evolution Specialist
####                    Jeff, Brand Logistics Specialist
####
####                    Pawnee
####                    Ron, Efficiency Logistics Strategist
####                    Leslie, Creative Evolution Strategist
####                    Ann, Data Evolution Engineer
####                    Mark, Operational Services Specialist
####                    Tom, Creative Logistics Coordinator
####                    April, Enterprise Services Consultant
####                    Andy, Creative Logistics Specialist
####                    Jerry, Enterprise Services Technician
####                    Donna, Operational Innovation Engineer
####
####                    Scranton
####                    Dwight, Efficiency Evolution Specialist
####                    Jim, Data Innovation Specialist
####                    Pam, Operational Analytics Technician
####                    Ryan, Data Evolution Consultant
####                    Stanley, Efficiency Logistics Technician
####                    Michael, Enterprise Communications Technician
####                    Kevin, Operational Services Technician
####                    Meredith, Data Evolution Technician
####                    Angela, Enterprise Communications Consultant
####                    Oscar, Creative Innovation Coordinator
####                    Phyllis, Enterprise Analytics Technician
####
####                    Ambiguous
####                    J.D., Efficiency Analytics Coordinator
####                    Turk, Efficiency Innovation Technician
####                    Elliot, Data Innovation Specialist
####                    Perry, Efficiency Evolution Technician
####                    Bob, Enterprise Logistics Engineer
####                    Carla, Brand Services Technician
####                    Janitor, Efficiency Evolution Coordinator

##print()
##hs.synergize()
##print()

##print(hs)
####                    Synergistic Management Solutions
```

```
####
####                    Pawnee
####                    Jerry, Enterprise Services Technician
####                    Ron, Efficiency Logistics Strategist
####                    Andy, Creative Logistics Specialist
####                    Leslie, Creative Evolution Strategist
####                    Ann, Data Evolution Engineer
####
####                    Scranton
####                    Dwight, Efficiency Evolution Specialist
####                    Jim, Data Innovation Specialist
####                    Pam, Operational Analytics Technician
####                    Ryan, Data Evolution Consultant
####                    Stanley, Efficiency Logistics Technician
####                    Michael, Enterprise Communications Technician
####                    Kevin, Operational Services Technician
####                    Meredith, Data Evolution Technician
####                    Angela, Enterprise Communications Consultant
####                    Oscar, Creative Innovation Coordinator
####                    Phyllis, Enterprise Analytics Technician
####
####                    Ambiguous
####                    J.D., Efficiency Analytics Coordinator
####                    Turk, Efficiency Innovation Technician
####                    Elliot, Data Innovation Specialist
####                    Perry, Efficiency Evolution Technician
####                    Bob, Enterprise Logistics Engineer
####                    Carla, Brand Services Technician
####                    Janitor, Efficiency Evolution Coordinator
####
####                    Greendale
####                    Ben, Enterprise Communications Consultant
####                    Craig, Enterprise Communications Engineer
####                    Britta, Enterprise Services Coordinator
####                    Pierce, Enterprise Evolution Specialist
####                    Jeff, Brand Logistics Specialist

##print()
##hs.synergize()
##print()

##print(hs)
####                    Scranton
####                    Meredith, Data Evolution Technician
####                    Kevin, Operational Services Technician
```

Phyllis, Enterprise Analytics Technician
Pam, Operational Analytics Technician
Michael, Enterprise Communications Technician
Stanley, Efficiency Logistics Technician
####
Ambiguous
J.D., Efficiency Analytics Coordinator
Turk, Efficiency Innovation Technician
Elliot, Data Innovation Specialist
Perry, Efficiency Evolution Technician
Bob, Enterprise Logistics Engineer
Carla, Brand Services Technician
Janitor, Efficiency Evolution Coordinator
####
Greendale
Ben, Enterprise Communications Consultant
Craig, Enterprise Communications Engineer
Britta, Enterprise Services Coordinator
Pierce, Enterprise Evolution Specialist
Jeff, Brand Logistics Specialist
####
Pawnee
Jerry, Enterprise Services Technician
Ron, Efficiency Logistics Strategist
Andy, Creative Logistics Specialist
Leslie, Creative Evolution Strategist
Ann, Data Evolution Engineer