**MSCF Python Programming Basics**

**Homework 5**

***Due At 11:59 pm US Eastern Daylight Time,***

***July 24, 2022***

1. **(100 points) Employees and Managers**
2. For this part, you will need the **Employee3.py** and **Manager.py** modules discussed during the second half of the Week 4 lecture (starting on slide 47), and the **hw5\_1.py** source code file posted with this homework.

In **hw5\_1.py**, you will see that part 1.a is written for you: it creates two **Employee** objects and two **Manager** objects, displays these four objects, then creates a **company** (a **list**) containing these four objects, and finally displays the items in the **company**.

Execute this code to confirm that it works as you would expect.

1. Parts 1.b and later are “commented out” using a pair of triple single quotes. Move the upper triple single quotes below 1.b and above 1.c.

What we want to do in part 1.b is to give raises to all of **company**’s **Employee**s (including **Manager**s: a **Manager** *is a kind of* **Employee**). But we want to give a 10% raise to each **Manager** and a 3% raise to each “ordinary” **Employee**.

Learn about the **isinstance()** function, and replace the commented pseudocode with working Python code. Then, test that your code for part 1.b works as you would expect.

1. You can enable equality and relational operator symbols (**==**, **!=**, **<**, **<=**, and so forth) to work with objects of a class by defining the special methods **\_\_eq\_\_()**, **\_\_ne\_\_()**, **\_\_lt\_\_()**, and so forth within the **Employee** class. Check the documentation here:

<https://docs.python.org/3/reference/datamodel.html?highlight=__eq__#object.__eq__>

Define these methods to compare **Employee** objects by **rate** (ignore the **name** and **id**). Once you have added these methods to your Employee class, move the upper triple single quotes below 1.c and above 1.d. Test that the code for part 1.c works as you would expect.

1. We use a *class variable* within **Employee** to count the number of **Employee** objects (including **Manager** objects, since a **Manager** is a kind of **Employee**) that have been created so far.

We would also like to know how many **Employee** objects *currently exist*. That is, if we create an **Employee** object, the number of **Employee** objects that have been created should be incremented, *and* the number of **Employee** objects that currently exist should also be incremented.

If we then use **del** to delete that **Employee** object, the number of **Employee** objects that have been created so far will remain unchanged, *but* the number of **Employee** objects that currently exist should be *decremented*.

If an object is created *locally* within a function definition, then that object will automatically be deleted when that function returns, even if we do not explicitly use a **del** statement. So in the **temp\_job()** function definition in part 1.d, four **Employee** objects are created when the function is called, and those four **Employee** objects are automatically deleted when the function returns.

Learn about the **\_\_del\_\_()** special method that automatically gets called when an object is deleted. Add a class variable to keep track of the number of **Employee** objects that currently exist. Add code to increment this variable within **\_\_init\_\_()** and to decrement this variable within **\_\_del\_\_()**. Add a **get** method that returns the value of this class variable.

Eliminate the triple single quotes that enclose the code for part 1.d, then test that the code for part 1.d works as you would expect.

***And Finally***

***REMEMBER*** to put all team members’ names into **hw5\_1.py**.Put your **hw5\_1.py**, **Employee3.py**, and **Manager.py** files into a **Team\_***N***\_HW5.zip** archive, where *N* is your team number, and upload to Canvas.