Lab 4: Vector Operation Lab

This is an INDIVIDUAL assignment. Due date is as indicated on BeachBoard. Follow ALL instructions otherwise you will lose points. In this lab, you will be overloading operators from two classes called `Vector` and `Matrix`.

Background:
You should already know how to do Vector addition, Vector subtraction, Dot products, Matrix addition, Matrix subtraction, and Matrix multiplication. If you do not know how to do this, please re-watch the lectures.
The other topic that you need to know is how to override class operators in Python. So, let's do a quick Python class/object review:

Defining a Class in Python:
Our goal is to make a class called Point. (x, y)
To define a class, use the keyword "class"

```
class Point:
    pass
```

This creates a class called Point. There is more that needs to be done though.

Constructors in Python:
A constructor is the function that is called whenever a new object is initialized.
** Note that class functions that begin and end with a double underscore are special functions

```
class Point:
    # default values of (0, 0)
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
```

The `self` parameter should be the first argument for all class functions. This is because `self` is used to represent the object. Let's dissect what is happening here:

- Line 3: `def __init__(self, x = 0, y = 0):`
  - `x` and `y` are given default values. If no arguments are given for the constructor, then they default to zero since `x=0` and `y=0`
- Line 4: `self.x = x`
  - `self.x` is a variable that belongs to the class
  - `x` is a local variable that is passed as an argument to the constructor
  - `self.x = x` will take the value that was passed as a function input and set it is the object's variable
  - Thus, there is a difference between `self.x` and x. This is why `self` is an important parameter to be passed

To create a new object

```
P1 = Point()
P2 = Point(-1, 3)
P3 = Point(y = 4)
```

```
print(P1)
print(P2)
print(P3)
```

In this example, we make three instances of Point:

- P1: takes advantage of default values. Should be (0, 0)
- P2: should be (-1, 3)
- P3: keeps default value of x = 0. should be (0, 4)

What is printed though, is the object address instead of the values that we saved.

Creating str function for debugging purposes:

```
class Point:
     # default values of (0, 0)
     def __init__(self, x = 0, y = 0):
          self.x = x
          self.y = y

     def __str__(self):
          string = '(' + str(self.x) + ', ' + str(self.y) + ')'
          return string

P1 = Point()
P2 = Point(-1, 3)
P3 = Point(y = 4)

print(P1)
print(P2)
print(P3)
```

- In the __str__ function
    - `self` is passed as a parameter because we want to access the object's `x` and `y` values. Not some local one. If we replace `self.x` and `self.y` with `x` and `y` respectively, then we'll get an error
    - We return the string that we want to see when we print the object
- Now we get the actual point because we have overridden the default __str__ function that just returns the object's address
-

Making object functions in Python:

You can make functions that are specific to the object.

```
class Point:
     # default values of (0, 0)
     def __init__(self, x = 0, y = 0):
          self.x = x
          self.y = y

     def distance(self):
          dist = (self.x**2 + self.y**2) ** (1/2)
          return dist
```

```
    def __str__(self):
        string = '(' + str(self.x) + ', ' + str(self.y) + ')\n'
        string += 'Distance: ' + str(self.distance())
        return string

P1 = Point()
P2 = Point(-1, 3)
P3 = Point(y = 4)

print(P1)
print(P2)
print(P3)
print('P1:', P1.distance())
print('P2:', P2.distance())
print('P3:', P3.distance())
```

Notice…

- In `distance()`
  - `self` parameter is still passed as an input
- In `__str__()`
  - To call the `distance()` function, we use `self.distance()` because this function is specific to the object
- In main
  - To call the function, we use `object_name.function_name()`
  - Note that we call this function differently (without `self`) because it is not within the function

Function overloading in Python:

Let's say that you have the following

```
class Point:

    # default values of (0, 0)
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

    def distance(self):
        dist = (self.x**2 + self.y**2) ** (1/2)
        return dist

    def __str__(self):
        string = '(' + str(self.x) + ', ' + str(self.y) + ')'
        # string += 'Distance: ' + str(self.distance())
        return string

P1 = Point()
P2 = Point(-1, 3)
P3 = Point(y = 4)
print(P1, '+', P2, '=', P1 + P2)
```

You would get an error because Python is unable to decipher how "+" should work when it has the two Point objects as arguments.
We must overload the operator by implementing __add__() in the class.

To overload other operators:

Operator	Expression	Function
Addition	`self + other`	`__add__(self, other)`
Subtraction	`self – other`	`__sub__(self, other)`
Multiplication	`self * other`	`__mul__(self, other)`

```
class Point:

     # default values of (0, 0)
     def __init__(self, x = 0, y = 0):
          self.x = x
          self.y = y

     def distance(self):
          dist = (self.x**2 + self.y**2) ** (1/2)
          return dist

     # This would work with the case: self + other (in this order)
     # This adds to the x/y attributes of self and other
     # Returns a new Point
     def __add__(self, other):
          x = self.x + other.x
          y = self.y + other.y
          return Point(x, y)

     def __str__(self):
          string = '(' + str(self.x) + ', ' + str(self.y) + ')'
          # string += 'Distance: ' + str(self.distance())
          return string

P1 = Point()
P2 = Point(-1, 3)
P3 = Point(y = 4)
print(P1, '+', P2, '=', P1 + P2)
```

No more error ☺

Instructions:

1. Take a close look at the `Vector.py` and `Matrix.py` files. There are some functions that have already been implemented. In addition, there are two operators that you need to overload: `__sub__` and `__mul__`. Read through both of their descriptions carefully.

File name	Functions completed	Functions to be completed

`Vector.py`	`Constructor`
	`get_ith_element`
	`__str__`
	`__add__`
		`__sub__`
		`__mul__`

`Matrix.py`	`Constructor`
	`get_element`
	`__str__`
		`__add__`
		`__sub__`
		`__mul__`

   READ THE DESCRIPTIONS CAREFULLY! You will lose points if you do not follow the instructions. We are using a grading script.
   ****Do NOT alter the already implemented functions in ANY significant way.

2. Your job is to implement the five highlighted functions/operators so that it passes any test case. There are four and six sample test cases provided for you in Vector.py and Matrix.py respectively, but these are not the only cases that we will test. We will be testing other test cases in the same way the test cases are presented. **DO NOT USE NUMPY FOR THE IMPLEMENTATION. If you do, then you will get a zero on this assignment!!!! The goal of the assignment is to implement the algorithms for each operation while proving your ability to use Python.**

3. After completing these functions, comment out the test cases (or delete them) or else the grading script will pick it up and mark your program as incorrect. Also, MAKE SURE THAT YOUR PROGRAM DOES NOT PRINT ANYTHING!!! The grading script may pick up any printed text and misinterpret it as a test case or input causing you to lose points.

4. Convert your `Vector.py` AND `Matrix.py` file to `.txt` files. Submit your `Vector.py`, `Matrix.py`, and your `.txt` files on BeachBoard. Do NOT submit it in compressed folder.

5. Do not email us your code asking us to verify it. We will answer general questions, but we will not debug your code over email.

Grading rubric

To achieve any points, your submission must have the following. Anything missing from this list will result in an automatic zero. NO EXCEPTIONS!

- Submit both py and txt files
- Files named correctly
- Program has no errors (infinite loops, syntax errors, logical errors, etc.) that terminates the program

Please note that if you change the function headers or if you do not return the proper outputs according to the function requirements, you risk losing all points for those test cases.

Points	Requirement
5	Submitted the correct 4 files: Vector.py, Matrix.py, Vector.txt, Matrix.txt
5	Passes 10 (4 from Vector and 6 from Matrix) original test cases. Also commented out or deleted the test cases
5	Implemented __sub__ in Vector correctly (Part 1)- 5 cases
5	Implemented __mul__ in Vector correctly (Part 2)- 5 cases
10	Implemented __add__ in Matrix correctly (Part 3)- 10 cases
10	Implemented __sub__ in Matrix correctly (Part 4) 10 cases
15	Implemented __mul__ in Matrix correctly (Part 5) – 10 cases

*** If you use numpy in this lab, you will get an automatic zero!