Lab 5: Object Oriented Programming

This assignment will involve figuring out how to use these new classes with only the lecture material to build off. A lot of programming involves figuring out how to use approaches or packages just from the documentation. I'm deliberately not providing a lot of information in this handout to try and give you practice gaining this slightly nebulous skill. If you feel stuck don't despair, use the documentation and online tutorials to figure it out. If you are still stuck after genuinely trying that to the best of your ability then the TAs are there to give you a hand.

Remember to submit your documented/clean notebook to brightspace.

Part 1: Creating a Vector class

There are some great python linear algebra packages (that we will use later in the course) but it is always good to understand roughly how operations work before trusting blindly to complex packages to do it for you. To serve this aim and to give you practice with classes, the following questions will involve you creating a class which can represent a vector.

We are only going to be focused on vectors of length 2 which represent a position in 2-dimensional space. You can just think of these as x and y coordinates in a standard cartesian plane.

Q1 [4 points]. Write a `Vector` class that allows you to define and create a simple 2-dimensional vector. This should:

- Have a clear docstring explaining what the class is and how it can be instantiated (created).
- Implement an `__init__` which takes two input values, checks whether they are valid floats or integers (and should cause an error using `raise ValueError`(MESSAGE) where MESSAGE is replaced with a string containing an appropriate error message) and assigns them to x and y attributes.
- Should implement a `__repr__` method which returns a string representation of the vector (this means print will work as expected).

```
>>> vector1  = Vector(5, 10)
>>> vector1.x == 5
True
>>> vector1.y == 10
True
>>> print(vector1)
Vector(5, 10)
>>> vector2 = Vector(1.2, -5.3)
>>> vector3 = Vector('a', 'z')
Error message explaining that x and y must be integers or floats
```

Q2 [2 points]. A vector's magnitude can be calculated using the following formula: $\sqrt{(x^2 + y^2)}$. Copy your code from the previous solution and add a method which returns the magnitude of a Vector object. You can get a function to calculate the square root by using python's built-in math module (https://docs.python.org/3/library/math.html)

```
>>> vector1 = Vector(5, 10)
>>> vector1.magnitude()
11.18033
>>> Vector(-1, -1).magnitude()
1.4142
>>> Vector(0, 0).magnitude()
0.0
```

Q3 [4 points]. Vector addition and subtraction works as follows:

$v1+v2=\langle v1_x + v2_x, v1_y + v2_y \rangle$

$v1-v2=\langle v1_x - v2_x, v1_y - v2_y \rangle$

Copy your code from the previous answers and add an `__add__` and `__sub__` method to your `Vector` class so that + and - operators work correctly. This should:

- Check whether the thing being added or subtracted to your Vector is also a Vector (using `type` (Note: this is only expected to work for Vector +/- other_thing, other_thing +/- Vector has other complications and will automatically throw a different error!)
- Return a new `Vector` with the correct values
- Include appropriate docstrings.

```
>>> Vector(-1, -1) + Vector(10, -4)
Vector(44,19)

>>> Vector(40, 10) - Vector(-4, -9)
Vector(44,19)

>>> Vector(13, 1) + 1
ValueError: Can only add Vector objects
>>> Vector(13, 1) - 1
ValueError: Can only subtract Vector objects
```

Q4 [4 points]. Vectors can be multiplied by a scalar (a single number) using the following formula:

v1 * scalar = (scalar * $v1_x$, scalar * $v1_y$).

Copy your solution to the previous question and add a `__mul__` method so you can multiply your vector by a scalar and return a new vector. You can implement the opposite ordered operation (i.e., scalar * vector) by defining the method `__rmul__` to return vector * scalar. Your code should check whether the scalar is a valid integer or float.

```
>>> Vector(-1, -1) * 5
Vector(-5,-5)

>>> 5 * Vector(-1, -1)
Vector(-5,-5)

>>> Vector(10, -4) * 0
Vector(0,0)

>>> Vector(-1, -1) * Vector(10, -4)
ValueError: Vector(10,-4) must be int or float
```

Q5 [3 points]. The dot product of 2 vectors is defined as

$v1 \cdot v2 = v1_x \times v2_x + v1_y \times v2_y$

Copy your solution from the previous question and modify `__mul__` so it also calculates the dot product if both inputs are vectors. Make sure you update the docstring appropriately and that scalar multiplication still works.

```
>>> Vector(-1, -1) * 5
Vector(-5,-5)

>>> 5 * Vector(-1, -1)
Vector(-5,-5)

>>> Vector(2,2) * Vector(-1, -1)
- 4

>>> Vector(2,7) * Vector(-4, 9)
55

>>> Vector(2,2) * 'a'
ValueError: a must be int or float
```

Q6 [4 points]. We can divide a vector by a scalar using the following formula:
$v1 / scalar = \langle v1_x / scalar, v1_y / scalar \rangle$.

We can also calculate the normalized vector by dividing it by its magnitude (i.e., a scalar):

$v1 / |v1| = \langle v1_x / |v|, v1_y / |v| \rangle$.

Copy your vector implementation and add a `__truediv__` method which divides a vector by a scalar (including checking for a valid int/float that is not 0).

Then implement a `normalise()` method which uses this to return the normalised vector (i.e., a new vector in the same direction but with magnitude 1).

Part 2: Inheritance

Classes can inherit attributes and methods from a specified parent class. We can also override those parent attributes and methods by reimplementing them in our child class. See the lecture slides from this week or the Python class documentation (https://docs.python.org/3/tutorial/classes.html#inheritance) for more explanation.

Q7 [8 points]. Using inheritance create a `BizzaroFloat` class that inherits from `float` and works as a normal float apart from when using special comparison methods. For each of the following reimplement them so BizzaroFloat returns the opposite bool value to the normal behavior (i.e., BizzaroFloat(5) == BizzaroFloat(10) returns True).

`x == y`	`x.__eq__(y)`
`x != y`	`x.__ne__(y)`
`<`	`__lt__`
`>`	`__gt__`

Part 3: Putting it all together

Q8 [8 points]. A video game has different types of characters, each with their own way of attacking other characters. Copy `GameCharacter` definition into your notebook, add docstrings, and implement `__repr__` which displays all the attribute values when you print a `GameCharacter` object.

Then define a `Warrior` and a `Mage` class which inherit from `GameCharacter` but also implement their own attack method. This `attack()` method should take another `Warrior` or `Mage` object as an argument and apply the appropriate attack damage to the health attribute of the other character. For the `Warrior` the `attack()` should deal 25 damage to the other character's health and reduce the warrior's own energy by 25. For `Mage` the `attack()` should decrease the other character's health by 40 and reduce the mage's own energy by 30.

Show your code works by creating a `Mage` and a `Warrior` and having them attack each other at least once then printing out their current health and energy levels.

```
class GameCharacter:
    def __init__(self, name, health=100):
        self.name = name
        self.health = health
        self.energy = 100

    def attack(self, other_character):
```

```
        pass

    def rest(self):
        recovered = min(25, 100 - self.energy)
        self.energy += recovered
        return recovered
```