6.101 Recitation 17: Week 10 Autocomplete Midpoint **4/17/24**

This sheet is yours to keep!

Question 1: Today we're going to build a planet physics simulation (see video.) Discuss with a neighbor: what classes would be a good idea to implement? What attributes and methods should each class have?

Question 2: Fill in the missing code below for the following Vector class methods:

```
class Vector:

    def __init__(self, coords):
       # each Vector object has a nd tuple of coords


    def __repr__(self):
        # repr(Vector([0, -4]))  -> 'Vector((0, -4))'



    def add(self, other):
        # Vector([1, 2]).add(Vector([1, 0]))  -> Vector((2, 2))



    def sub(self, other):
        # Vector([1, 2]).sub(Vector([1, 0]))  -> Vector((0, 2))



    def mul(self, other):
        # Vector([1, 2]).mul(5)   -> Vector((5, 10))



    def div(self, other):
        # Vector((4, 2)).div(2)   -> Vector((2.0, 1.0))



    def magnitude(self):
        # Vector((3, 4)).magnitude() -> 5.0



    def normalize(self):  # creates a unit vector in the same direction
        # Vector([3, 4]).normalize()   -> Vector([.6, .8])
```

R17 Participation Credit **Kerberos : _____@mit.edu**

Hand this sheet in at the end of recitation to get participation credit for today.

Question 4: Now that we have rewritten the Vector class in terms of dunder methods such as __add__, __sub__, __mul__, etc. (see Question 3), edit the Body class below to make use of these new Vector methods. What do you notice about the resulting code?

```
class Body:
    def __init__(self, mass, position, initial_velocity=None):
        self.mass = mass
        self.position = position
        self.velocity = initial_velocity or Vector((0, 0))


    def force_from(self, other):

        delta = other.position.sub(self.position)

        dist = delta.magnitude()

        direction = delta.normalize()

        magnitude = (GRAVITATIONAL_CONSTANT * self.mass * other.mass) / (dist * dist)

        return direction.scale(magnitude)



    def net_force(self, bodies):
        # F = sum of G * m1 m2 / |r|^2  * r^
        F = Vector([0,0])

        for other in bodies:

            if other is self:  # note the is keyword!

                continue

            F = F.add(self.force_from(other))

        return F



    def move(self, f, dt):

        acceleration = f.div(self.mass)

        self.velocity = self.velocity.add(acceleration.scale(dt))

        self.position = self.position.add(self.velocity.scale(dt))
```

Question 3: Fill in the missing code below for the following Vector class methods:

```
class Vector:
    """
    Nd Vector object; has immutable tuple of coords
    """

    def __init__(self, coords):



    def __repr__(self):
        # repr(Vector([0, -4])) -> 'Vector((0, -4))'




    def __add__(self, other):
        # Vector([1, 2]) + Vector([1, 0]) -> Vector((2, 2))




    def __sub__(self, other):
        # Vector([1, 2]) - Vector([1, 0]) -> Vector((0, 2))




    def __mul__(self, other):
        # Vector([1, 2]) * 5 -> Vector((5, 10))



    def __truediv__(self, other):
        # Vector([4, 2]) / 2 ->  Vector((2.0, 1.0))



    def __abs__(self):
        # abs(Vector([3, 4]))  -> 5.0



    def normalize(self):
        # Vector([3, 4]).normalize() -> Vector((.6, .8))
```