Entry format example

Date: 10/2/2024

Description: Example

Tests done: Example

Code: Example

Notes: Any

|  |
| --- |
| Date: 3/3/2024 |
| Description:   Defined vector class and methods.  Defined planet class and methods.  Vector class has addition, subtraction, multiplication and division (division only with constants)  Planet class has a way of adding forces up for resultant force and then finding acceleration by dividing it with the mass of the planet using the method defined in the vector class. |
| Tests done:   I have tested if the planet class is working using its methods to find the resultant force and then finding the acceleration.   Running the test python code below gives this output: |
| Test python code:   |  | | --- | | from my\_classes import vector  from my\_classes import planet  planet\_list = []  earth = planet('earth', 5.972\*10\*\*24, vector(0,0), vector(10,10), vector(0,5))  jupiter = planet('jupiter', 2\*10\*\*27, vector(1000,0), vector(30,5), vector(5,1))  planet\_list = [earth,jupiter]  acceleration = earth.find\_acceleration(planet\_list,6.67430\*10\*\*-11)  print(acceleration.x) # fix this  print(acceleration.y) | |
| Classes are written here   |  | | --- | | planet\_list = []  class planet:  def \_\_init\_\_(self,name,mass,position,velocity,acceleration): # use vectors for position, velocity and acceleration  self.name = name  self.mass = mass  self.position = position  self.velocity = velocity  self.acceleration = acceleration  def distance\_from(self,planet):  position\_vector1 = self.position  position\_vector2 = planet.position  distance = position\_vector1.distance\_to(position\_vector2)  return distance  def resultant\_force(self, planet\_list, gravitational\_constant):  resultant\_force = vector(0,0)  for i in planet\_list:  if i != self:  unit\_vector = self.position.unit\_vector\_to(i.position)  mass\_product = self.mass \* i.mass  distance\_squared = self.distance\_from(i)\*\*2  multiple = -gravitational\_constant \* mass\_product / distance\_squared  force = unit\_vector.multiply\_by(multiple) # using vector form of newton's law of gravitation. this is rearranged so the unit vector is in front  resultant\_force = resultant\_force.add(force)  return resultant\_force  def find\_acceleration(self,planet\_list,gravitational\_constant):  resultant\_force = self.resultant\_force(planet\_list,gravitational\_constant)  acceleration = resultant\_force.divide\_by(self.mass)  return acceleration  class vector:  def \_\_init\_\_(self,x,y):  self.x=x  self.y=y  def multiply(self,vector):  x = self.x \* vector.x  y = self.y \* vector.y  return x + y  def print\_position(self):  print(self.x)  print(self.y)  def get\_position(self):  return [self.x,self.y]  def magnitude(self):  magnitude\_ = (self.x\*\*2 + self.y\*\*2)\*\*0.5  return magnitude\_  def unit\_vector\_to(self,vector1):  displacement\_vector = vector1.sub(self)  distance = displacement\_vector.magnitude()  unit\_vector = displacement\_vector.divide\_by(distance)  return unit\_vector    def add(self,vector1):  x = self.x + vector1.x  y = self.y + vector1.y  return vector(x,y)  vector.add = add  def sub(self,vector1):  x = self.x - vector1.x  y = self.y - vector1.y  return vector(x,y)  vector.sub = sub  def distance\_to(self,vector1):  diff\_vector = self.sub(vector1)  return diff\_vector.magnitude()  vector.distance\_to = distance\_to  def divide\_by\_constant(self,constant):  x = self.x/constant  y = self.y/constant  return vector(x,y)  vector.divide\_by=divide\_by\_constant  def multiply\_by\_constant(self,constant):  x = self.x \* constant  y = self.y \* constant  return vector(x,y)  vector.multiply\_by = multiply\_by\_constant | |
| Notes:   Acceleration does not give a correct value. The output was 133486000000.0 ms^-2 which does not make sense in a physical situation. Acceleration is much more likely to be something like 1.2 ms^-2 instead. Check if the distance between Earth and Jupiter is sufficient. |