|  |
| --- |
| Date: 5/3/2024 |
| Description:   More thorough testing of the vector class. |
| Tests Done:   Created a vector called my\_vector and another vector called my\_vector2.  Used the vector.add method defined in the vector class to add them together.  my\_vector is the vector (10,5) and my\_vector2 is the vector (100,3)  Adding them should result in a vector (110,8) and the code correctly adds them together.  Here is a testing table:   |  |  |  |  | | --- | --- | --- | --- | | my\_vector | my\_vector2 | Expected | Output | | (10,5) | (100,3) | (110,8) | (110,8) | | (45,9) | (70,2) | (115,11) | (115,11) | | (1230,6) | (44,1) | (1274,7) | (1274,7) |   Used the vector.multiply method defined in the vector class to multiply two vectors together.  Here is a testing table:   |  |  |  |  | | --- | --- | --- | --- | | my\_vector | my\_vector2 | Expected | Output | | (3,4) | (1,2) | 11 | 11 | | (10,7) | (3,4) | 58 | 58 | |
| Testing Code:  For testing .add method   |  | | --- | | from my\_classes import vector  from my\_classes import planet  my\_vector = vector(1230,6)  my\_vector2 = vector(44,1)  resultant\_vector = my\_vector.add(my\_vector2) # uses the .add method  resultant\_vector.print\_position() # outputs to check if it adds correctly. |   For testing .multiply method   |  | | --- | | from my\_classes import vector  from my\_classes import planet  my\_vector = vector(10,7)  my\_vector2 = vector(3,4)  result = my\_vector.multiply(my\_vector2)  print(result) # should return 11 | |
| Classes Code   |  | | --- | | 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  #-----------------------------------------------------------------------------------------------END OF PLANET CLASS  class vector:  def \_\_init\_\_(self,x,y):  self.x=x  self.y=y  def multiply(self,vector): # tested on 5/3/2024 - refer to log 2  x = self.x \* vector.x  y = self.y \* vector.y  return x + y  def print\_position(self): # tested on 5/3/2024 - refer to log 2  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): # tested on 5/3/2024 - refer to log 2  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  #-----------------------------------------------------------------------------------------------END OF VECTOR CLASS  class integration\_term:  def \_\_init\_\_(self,coefficient,power):  self.coefficient = coefficient  self.power = power  def integrate(self): # this ignores constant of integration which needs to be added at the end after the whole expression is evaluated  self.power = self.power + 1 # add one to the power  self.coefficient = self.coefficient / self.power # divide by the new power  return self  class integration\_expression:  def \_\_init\_\_(self,\*terms):  self.terms = terms  def integrate(self):  new\_terms = []  for term in self.terms:  new\_term = term.integrate()  new\_terms.append(new\_term)  self.terms = new\_terms  return self.terms | |