Report

My team is myself(I decide to do this on my own and this assignment is all done by myself). The assignment really helped me learned about python class(no inheritance), function, variable, module. The assignment is easier than I thought but there are still challenges.

I first did the assignment without operator overloading and it appears that it is really easy. However during class professor told me to include operator overloading because it is required.

Then I meet two challenges.

First challenge is that, since I have to use operator overloading, where do I find operators that I can overload? I start by google “python += overloading” etc. Then I find a python official website that records all the function that could be overloaded. Specifically speaking, the operators in python is not just a char. For example, a \* b would be a.\_\_mul\_\_(b). Now it is very easy to understand the overloading method. And it would be easy to find others.

Then challenge is overloading *=, I use \_\_idiv\_\_. However there are always errors. Then I looked at python official document and it turned out that the the website I used is for old versions. Newer version doesn’t use and \_\_idiv\_\_ anymore. Instead, it uses \_\_ifloordiv\_\_ \_\_itruediv\_\_. I made a silly mistake that when I call = or , I use a = b or a/b where b is a vec3d object. It complains that operands doesn’t match. I take a long time to look at how I implement the function.*

*The biggest challenge is overloading \*. In the example c++ file, there are actually 2 \* overloading. One is for dot product which return a double. Another is for scalar multiply which return a vec3d object. I have no idea how to implement that. I check online and they say the later function will make previous function useless. That means that if you have 2 functions with same name and parameter, the first function is regarded as garbage. So there is no way to do the same as the c++ file do which is making overloading functions. Then I figured that the only difference would be that in dot product, the parameter is a vec3d while in scalar multiply, the parameter is a double. So my goal is to make a if-else statement that would distinguish 2 situations. If a \* b and b is a number, execute one block of code, if not, execute another block of code. Then I meet another big challenge.*

*My initial version:*

*if type(a) is ‘float’ or ‘int’:*

*…*

*else:*

*…*

*However, the strange thing is that no matter what a is, it always falls into the if. I checked what the datatype of parameter is in test program. It is always different. One is number, one is vec3d.*

*I have no idea what happens. In test program, the datatype is different, but in code, it never falls into else statement.*

*I tried so many ways to solve this but all failed. I used rmul instead of mul. I tried both mul and rmul. I used switch statement instead of if else. I tried to add another parameter to \_\_mul\_\_. All failed.*

*Then I remember that when searching google, I saw a function that is isinstance. It also checks type. So I tried. It worked.*

*Even until now I have no idea why type() doesn’t work but isinstance works.*

*There is also a small challenge. How do I check if answers are right. I forget all the cross product, norm, self scale….. I went back and reviewed a little to finish this.*

*I learned these mainly from stackoverlow and python official document.*

*After finishing this, I don’t like python anymore. Since I spend almost 2 hours on figuring out how to overload \*, I feel like c++ is so friendly. You declare data for parameter. You need return type. Everything is so clear. However in python, you have no idea what the datatype of one parameter is. It could be anything.*

*In my test program, for easiness, I write a function that prints out my 2 example vec3d. I can literally write a,x b,x even though the program does not know what a is. It doesn’t know if a has an attribute x.*

*This is not looking good for me.*

PROGRAM

*from \_\_future\_\_ import division*

*import math*

*import sys*

*def deg\_to\_rad(a):*

*return a \* math.pi / 180*

*def rad\_to\_deg(a):*

*return a \* 180 / math.pi*

*class Vector3D:*

*##consructor*

*def \_\_init\_\_(self, xx, yy, zz):*

*self.x = xx*

*self.y = yy*

*self.z = zz*

*def print\_vec(self):*

*print(self.x, self.y , self.z)*

*def assign\_val(self, a, b, c):*

*self.x = a*

*self.y = b*

*self.z = c*

*return self*

*def assign\_vec(self, a):*

*self.x = a.x*

*self.y = a.y*

*self.z = a.z*

*def \_\_eq\_\_(self, a):*

*return self.x == a.x and self.y == a.y and self.z == a.z*

*def \_\_iadd\_\_(self, a):*

*self.x += a.x*

*self.y += a.y*

*self.z += a.z*

*return self*

*def \_\_isub\_\_(self, a):*

*self.x -= a.x*

*self.y -= a.y*

*self.z -= a.z*

*return self*

*def \_\_imul\_\_(self, a : float):*

*self.x \*= a*

*self.y \*= a*

*self.z \*= a*

*return self*

*def \_\_itruediv\_\_(self, a : float):*

*self.x /= a*

*self.y /= a*

*self.z /= a*

*return self*

*def \_\_ixor\_\_(self, a):*

*self.x \*= a.x*

*self.y \*= a.y*

*self.z \*= a.z*

*return self*

*def \_\_imod\_\_(self, a):*

*xx = self.x*

*yy = self.y*

*zz = self.z*

*self.x = yy \* a.z - zz \* a.y*

*self.y = zz \* a.x - xx \* a.z*

*self.z = xx \* a.y - yy \* a.x*

*return self*

*def \_\_neg\_\_(self):*

*return Vector3D(-self.x, -self.y, -self.z)*

*def \_\_add\_\_(self, a):*

*return Vector3D(self.x + a.x, self.y + a.y, self.z + a.z)*

*def \_\_sub\_\_(self, a):*

*return Vector3D(self.x - a.x, self.y - a.y, self.z - a.z)*

*def \_\_mul\_\_(self, a):*

*if isinstance(a, int) or isinstance(a, float):*

*return Vector3D(self.x \* a, self.y \* a, self.z \* a)*

*else:*

*return (self.x \* a.x + self.y \* a.y + self.z \* a.z)*

*def \_\_truediv\_\_(self, a):*

*return Vector3D(self.x / a, self.y / a, self.z / a)*

*def \_\_xor\_\_(self, a):*

*return Vector3D(self.x \* a.x, self.y \* a.y, self.z \* a.z)*

*def \_\_mod\_\_(self, a):*

*b = Vector3D(self.x, self.y, self.z)*

*b %= a*

*return b*

*def norm(self):*

*a = self.normsqr()*

*return math.sqrt(a)*

*def normsqr(self):*

*return self \* self*

*def selfNormalize(self):*

*a = self.norm()*

*if a < sys.float\_info.epsilon:*

*self = Vector3D(0, 0, 0)*

*return self*

*else:*

*self /= a*

*return self*

*def normalize(self):*

*a = self.norm()*

*if a < sys.float\_info.epsilon:*

*self = Vector3D(0, 0, 0)*

*return self*

*else:*

*return self / a*

*def comp(self, a):*

*return self \* (a.normalize())*

*def self\_scale(self, a):*

*b = self.norm()*

*if b < sys.float\_info.epsilon:*

*self = Vector3D(0, 0, 0)*

*return self*

*else:*

*self \*= (a / b)*

*return self*

*def scale(self, a):*

*b = self.norm()*

*if b < sys.float\_info.epsilon:*

*self = Vector3D(0, 0, 0)*

*return self*

*else:*

*return self \* (a/b)*

*def rotateX(self, a):*

*c = math.cos(a)*

*s = math.sin(a)*

*return self.assign\_val(self.x, self.y \* c - self.z \* s, self.y \* s + self.z \* c)*

*def rotateXd(self, a):*

*return self.rotateX(deg\_to\_rad(a))*

*def rotateY(self, a):*

*c = math.cos(a)*

*s = math.sin(a)*

*return self.assign\_val(self.x \* c + self.z \* s, self.y, -self.x \* s + self.z \* c)*

*def rotateYd(self, a):*

*return self.rotateY(deg\_to\_rad(a))*

*def rotateZ(self, a):*

*c = math.cos(a)*

*s = math.sin(a)*

*return self.assign\_val(self.x \* c - self.y \* s, self.x \* s + self.y \* c, self.z)*

*def rotateZd(self, a):*

*return self.rotateZ(deg\_to\_rad(a))*

*def reset(self):*

*self.x = 0*

*self.y = 0*

*self.z = 0*

*def GetX(self):*

*return self.x*

*def GetY(self):*

*return self.y*

*def GetZ(self):*

*return self.z*

*def SetX(self, a):*

*self.x = a*

*def SetY(self, a):*

*self.y = a*

*def SetZ(self, a):*

*self.z = a*

*def Set\_All(self, a, b, c):*

*self.x = a*

*self.y = b*

*self.z = c*

*TEST PROGRAM*

import Vector

#########################################################

a = Vector.Vector3D(1, 2, 3)

b = Vector.Vector3D(4, 5, 6)

def print\_a\_b():

print("a is ", a.x, a.y, a.z)

print("b is ", b.x, b.y, b.z)

print\_a\_b()

print(" ")

print("assigning 3 values to a (5 6 7):")

a.assign\_val(5, 6, 7)

print\_a\_b()

print("is a and b equal?????")

print(a == b)

print(" ")

print("now letting a equal to b:")

a.assign\_vec(b)

print\_a\_b()

print(" ")

print("is a and b equal?")

print(a == b)

print(" ")

print("a += b")

a += b

print\_a\_b()

print(" ")

print("a -= b")

a -= b

print\_a\_b()

print(" ")

print("a \*= 2")

a \*= 2

print\_a\_b()

print(" ")

print("a /= 2")

a /= 2

print\_a\_b()

print(" ")

print("c = -a")

c = -a

print(c.print\_vec())

print(" ")

print("now give a a smaller value for easy testing")

a.assign\_val(1, 2, 3)

print\_a\_b()

print(" ")

print("a ^= b")

a ^= b

print\_a\_b()

print(" ")

print("now give a a smaller value for easy testing")

a.assign\_val(1, 2, 3)

print\_a\_b()

print(" ")

print("a %= b")

a %= b

print\_a\_b()

print(" ")

print("now give a a smaller value for easy testing")

a.assign\_val(1, 2, 3)

print\_a\_b()

print(" ")

print("c = a + b")

c = a + b

c.print\_vec()

print(" ")

print("c = a - b")

c = a - b

c.print\_vec()

print(" ")

print("c = a \* 2")

c = a \* 2

c.print\_vec()

print(" ")

print("c = a / 2")

c = a / 2

c.print\_vec()

print(" ")

print("c = a ^ b")

c = a ^ b

c.print\_vec()

print(" ")

print("c = a % b")

c = a % b

c.print\_vec()

print(" ")

print("c = a \* b")

c = a \* b

print(c)

print(" ")

print("normsqr(a)")

print(a.normsqr())

print(" ")

print("norm(a)")

print(a.norm())

print(" ")

print("a.selfNormalize()")

a.selfNormalize()

a.print\_vec()

print(" ")

print("now give a a smaller value for easy testing")

a.assign\_val(1, 2, 3)

print\_a\_b()

print(" ")

print("c = a.normalize()")

c = a.normalize()

c.print\_vec()

print(" ")

print("c = a.comp(b)")

c = a.comp(b)

print(c)

print(" ")

print("a.self\_scale(5)")

a.self\_scale(5)

a.print\_vec()

print(" ")

print("now give a a smaller value for easy testing")

a.assign\_val(1, 2, 3)

print\_a\_b()

print(" ")

print("c = a.scale(5)")

c = a.scale(5)

c.print\_vec()

print(" ")

print("c = a.rotateXd(180)")

c = a.rotateXd(180)

c.print\_vec()

print(" ")

print("c = a.rotateYd(180)")

c = a.rotateYd(180)

c.print\_vec()

print(" ")

print("c = a.rotateZd(180)")

c = a.rotateZd(180)

c.print\_vec()

print(" ")

OUTPUT

jiazhen@jiazhen-MS-7B98:~/Desktop/programming language concepts/ball/py$ python3 test.py

a is 1 2 3

b is 4 5 6

assigning 3 values to a (5 6 7):

a is 5 6 7

b is 4 5 6

is a and b equal?????

False

now letting a equal to b:

a is 4 5 6

b is 4 5 6

is a and b equal?

True

a += b

a is 8 10 12

b is 4 5 6

a -= b

a is 4 5 6

b is 4 5 6

a \*= 2

a is 8 10 12

b is 4 5 6

a /= 2

a is 4.0 5.0 6.0

b is 4 5 6

c = -a

-4.0 -5.0 -6.0

None

now give a a smaller value for easy testing

a is 1 2 3

b is 4 5 6

a ^= b

a is 4 10 18

b is 4 5 6

now give a a smaller value for easy testing

a is 1 2 3

b is 4 5 6

a %= b

a is -3 6 -3

b is 4 5 6

now give a a smaller value for easy testing

a is 1 2 3

b is 4 5 6

c = a + b

5 7 9

c = a - b

-3 -3 -3

c = a \* 2

2 4 6

c = a / 2

0.5 1.0 1.5

c = a ^ b

4 10 18

c = a % b

-3 6 -3

c = a \* b

32

normsqr(a)

14

norm(a)

3.7416573867739413

a.selfNormalize()

0.2672612419124244 0.5345224838248488 0.8017837257372732

now give a a smaller value for easy testing

a is 1 2 3

b is 4 5 6

c = a.normalize()

0.2672612419124244 0.5345224838248488 0.8017837257372732

c = a.comp(b)

3.6467384467084143

a.self\_scale(5)

1.3363062095621219 2.6726124191242437 4.008918628686366

now give a a smaller value for easy testing

a is 1 2 3

b is 4 5 6

c = a.scale(5)

1.3363062095621219 2.6726124191242437 4.008918628686366

c = a.rotateXd(180)

1 -2.0000000000000004 -2.9999999999999996

c = a.rotateYd(180)

-1.0000000000000004 -2.0000000000000004 2.9999999999999996

c = a.rotateZd(180)

1.0000000000000007 2.0000000000000004 2.9999999999999996