Assignment 2 report

Amit Binu

binua

A Code for pointADT.py

```
## @file pointADT.py

## @title pointADT
# @author Amit Binu
# @date 19/2/2017
from math import *
## @brief Creates a point by taking its xcoordinate and ycoordinate
# @details This class has 4 methods. 2 of them are getters, 1 of them is a modifier and the other one
    returns the distance between 2 points.
class PointT(object):

    ## @breif PointT's constructor
    # @details The constructor assigns the values of xccordinate and ycoordinate to the varibales.
    # @param xc is the xcoordinate of the point
    # @param yc is the ycordinate of the point
    def __init__(self,x,y):
        self.xc = x
        self.yc = y

    ## @brief Returns the xcoordinate of the point
    # @return an integer value of the point's xcoordinate
    def xcrd(self):
        return self.xc

    ## @brief Returns the ycoordinate of the point
    # @return an integer value of the point's ycoordinate
    def ycrd(self):
        return self.yc
    ## @brief Returns the distance between two points
    # @details This method uses Pythagoream theorem to do the math.
    # @param p is a point object which has its own xcoordinate nad ycoordinate
    # @return an integer value of the distance between 2 points
    def dist(self,p):
        return sqrt((self.xc  - p.xcrd())**2 + (self.yc - p.ycrd())**2)

    ## @brief Changes the xcoordinate and ycoordinate of the point
    # @param theta is the shift value that is used to change point's xcoordinate and ycoordinate
    def rot(self, theta):
        t = (round(cos(theta))*self.xc ) - (round(sin(theta)) * self.yc)
        t1 = (round(sin(theta))*self.xc ) + (round(cos(theta)) * self.yc)
        self.xc = t
        self.yc = t1
```

B Code for LineADT

```
## @file lineADT.py

## @title lineADT
# @author Amit Binu
# @date 19/2/2017
import pointADT


b = pointADT.PointT(5,4)
e = pointADT.PointT(2,3)

## @brief Creates a line by taking two point objects
# @details This class has 5 methods. 2 of them are getters, 1 of them is a modifier and the others
    represent some properties of the line.
class LineT(object):

    ## @breif The constructor assigns the 2 pint objects to the self varibales.
    # @param b is the first point object.
    # @param e is the second point object.
    def __init__(self,p1,p2):
        self.b = p1
        self.e = p2

    ## @brief This is a getter method.
    # @return the first point that is created to make a line.
    def beg(self):
        return self.b

    ## @brief This is a getter method.
    # @return the second point that is created to make a line.
    def end(self):
        return self.e

    ## @brief This method returns the distance between these poitns.
    # @return an integer value that represents the distance between points b and e that are taken as
        parameters for the class LinetT.
    def len(self):
        return (self.b).dist(self.e)

    ## @brief This method calcuates and returns the mid point of the line created between points b and e.
    # @return a point object that will have the xcoordinate and ycoordinate of the mid point of the line
        between points b and e.
    def mdpt(self):
        if (self.len() == 0):
            return pointADT.PointT(0,0)

        return pointADT.PointT(  avg((self.b).xcrd(),(self.e).xcrd()),
            avg((self.b).ycrd(),(self.e).ycrd())  )

    ## @brief This method changes the values of xcoordinate and ycoordinate of the points b and e.
    # @param theta is the value that is used to change the xcoordinate and ycoordinate values of b and e.
    def rot(theta):
        (self.b).rot(theta)
        (self.e).rot(theta)

## @brief A local function
# @details This function does the average of two numbers
# @param x1 is a number
# @param x2 is another number
# @return a value that represents the average of the two numbers that were taken as parameters.
def avg(x1,x2):
    return (x1+x2)/2.0


j = LineT(b,e)
```

C Code for CircleADT.py

```
import lineADT
import pointADT

from math import *


## @file circleADT.py
#  @author Amit Binu
#  @brief Has a class that creates 2d circles
#  @date 19/02/2017

## @brief Creates a point by taking its xcoordinate and ycoordinate
# @details This class has 6 methods. 2 of them are getters and the others represent some properties of
    the circle.
class CircleT(object):

    ## @brief Constructor for the circleADT
    # @param cin is a point object that represents the midpoint of the circle
    # @param rin is an integer value that represents the radius of the circle
    def __init__(self, cin, rin):
        self.c = cin
        self.r = rin

    ## @brief This is a getter method
    # @return a point object that represents the middle point of the circle.
    def cen(self):
        return self.c

    ## @breif This is a getter method
    # @return an integer value that represents the radius of the circle.
    def rad(self):
        return self.r

    ## @brief This method calculates the area of the circle using the Pi from math libraray.
    # @return an integer value that represents the area of the cirlce
    def area(self):
        return pi * ((self.r)**2.0)

    ## @brief Checks whether two circles interesect or not.
    # @details p is a point object whose coordinates are average of the coordinates of 2 Circles'
        centeres.
    # @details It is assumed that the if one circle is inside the other, then it intersects.
    # @param ci is a CircleT object, so it is like another circle.
    # @return True if the 2 circle objects intersect and Fasle is it doesn't.
    def intersect(self, ci):
        p = pointADT.PointT(( (self.c).xcrd() + (ci.cen()).xcrd()) /2.0,( (self.c).ycrd() +
            (ci.cen()).ycrd()) /2.0 )

        return insideCircle(p, self) and insideCircle(p, ci)

    ## @breif makes a new connection with a new circle object
    # @return a line object that represents the line between the nid points of the 2 circle object. This
        line represents the connection between these 2 circle objects.
    def connection(self, ci):
        return lineADT.LineT(self.c, ci.cen())

    ## @breif Calculates the force that one circle expereinces from the other circle
    # @details This method uses the formula for the gravitational force. However, instead of the mass
        area() is used to calculate this.
    # @details Gravitational force is the force between 2 planets/2electrons. In this module, it is the
        force between 2 circles.
    # @param f is a function that has the equation for gravitational force.
    # @return The value of the force between two circles.
    def force(self, f):
        return (lambda x: (self.area()) * (x.area()) * f((self.connection(x)).len()))

# @brief A local function that checks whether the distance between a point and a circle's center is less
    than or equal to the circle's radius.
# @details This function uses cen() and rad() methods to get the circle's center point and circle's
    radius.
# @param p is a point object
# @param c is a circle object
# @ return a boolean value. True if the distance between a point and a circle's center is less than ot
    equal to the circle's radius and False otherwise.
def insideCircle(p,c):
    return p.dist(c.cen()) <= c.rad()
```

D Code for Deque.py

```
## @file deque.py

## @title deque
# @author Amit Binu
# @date 19/2/2017
from circleADT import *
from math import *


## @brief This is an abstract object that represents a queue that has a sequence of circles.
class Deq:

    MAX_SIZE = 20
    s = []

    def init():
        Deq.s = []

    @staticmethod
    def pushBack(c):
        if (len(Deq.s) <= Deq.MAX_SIZE):

            (Deq.s).append(c)


        else:
            raise FULL("The maximum size of the queue is 20 !")

    @staticmethod
    def pushFront(c):
        if (len(Deq.s) <= Deq.MAX_SIZE):
                (Deq.s).insert(0,c)

        else:
            raise FULL("The maximum size of the queue is 20 !")

    @staticmethod
    def popBack():
        if (len(Deq.s) != 0):
            return (Deq.s).pop(len(Deq.s) - 1)
        else:
            raise EMPTY("The queue is empty!")

    @staticmethod
    def popFront():
        if (len(Deq.s) != 0):
            return (Deq.s).pop(0)

        else:
            raise EMPTY("The queue is empty!")

    @staticmethod
    def back():
        if (len(Deq.s) != 0):
            return (Deq.s)[len(Deq.s) - 1]
        else:
            raise EMPTY("The queue is empty!")

    @staticmethod
    def front():
        if (len(Deq.s) != 0):
            return (Deq.s)[0]
        else:
            raise EMPTY("The queue is empty!")

    @staticmethod
    def size():
        return (len(Deq.s))

    @staticmethod
    def disjoint():
        if (len(Deq.s) == 0):
            raise EMPTY("The queue is empty")
        else:
            for i in range (len(Deq.s)):
                for j in range (len(Deq.s)):
```

```
                    if i != j:
                        if ((Deq.s)[i].intersect((Deq.s)[j])):
                            return False

            return True

    @staticmethod
    def sumFx(f):
        if (len(Deq.s) == 0):
            raise EMPTY("The queue is empty")
        else:
            sum = 0
            for i in range (len(Deq.s)):
                sum = sum  +  Fx(f, Deq.s[i], Deq.s[0])

            return sum

    @staticmethod
    def totalArea():
        size = len(Deq.s)
        if size == 0:
            raise EMPTY("The queue is empty")

        else:
            sum_of_area = 0
            for i in range(len(Deq.s)):
                sum_of_area = sum_of_area + ((Deq.s)[i]).area()

            return sum_of_area


    @staticmethod
    def averageRadius():
        size = len(Deq.s)
        if size == 0:
            raise EMPTY("The queue is empty")
        else:
            average_radius =0

            for i in range(len(Deq.s)):
                average_radius = average_radius + ((Deq.s)[i].rad())

            return average_radius / (1.0* (len(Deq.s)))


class FULL(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return str(self.value)

class EMPTY(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return str(self.value)

def Fx(f, ci, cj):
    return xcomp(ci.force(f)(cj), ci, cj)

def xcomp(F, ci, cj):
    return F*((((ci.cen()).xcrd()) - ((cj.cen()).xcrd()))/((ci.connection(cj)).len()))
```

E Code for testCircleDeque

```
import unittest
from math import *
from lineADT import *
from circleADT import *
from pointADT import *
from deque import *

class __main__(unittest.TestCase):
    def test_xcrd_are_equal(self):
        point = PointT(5,2)
        self.assertTrue(point.xcrd() == 5)

    def test_ycrd_are_equal(self):
        point = PointT(5,2)
        self.assertTrue(point.ycrd() == 2)

    def test_dist_are_equal(self):
        point = PointT(5,2)
        point2 = PointT(2,2)
        self.assertTrue(point.dist(point2) == 3)

    def test_rot_for_positive_value(self):
        point = PointT(5,2)
        point.rot(pi/2)
        self.assertTrue((point.xcrd()==-2.0) and (point.ycrd()==5.0))

    def test_rot_for_negative_value(self):
        point = PointT(20,24)
        point.rot(-pi)
        self.assertTrue((point.xcrd()==-20.0) and (point.ycrd()==-24.0) )


    def test_rot_for_zero(self):
        point = PointT(50,21)
        point.rot(0)
        self.assertTrue(point.xcrd() == 50 and point.ycrd() == 21)

    def test_rot_for_theta_greater_than_pi(self):
        point = PointT(6,4)
        point.rot(3*pi/2)
        self.assertTrue(point.xcrd() == 4 and point.ycrd() == -6)

    def test_beg_in_lineADT(self):
        point = PointT(5,2)
        point2= PointT(1,1)
        line = LineT(point, point2)
        self.assertTrue((line.beg()).xcrd() == 5 and (line.beg()).ycrd() == 2)

    def test_end(self):
        point = PointT(5,2)
        point2= PointT(1,1)
        line = LineT(point, point2)
        self.assertTrue((line.end()).xcrd() == 1 and (line.end()).ycrd() == 1)

    def test_len(self):
        point = PointT(5,0)
        point2= PointT(1,0)
        line = LineT(point, point2)
        self.assertTrue((line.len()) == 4)

    def test_mdpt(self):
        point = PointT(5,0)
        point2= PointT(1,0)
        line = LineT(point, point2)
        self.assertTrue((line.mdpt()).xcrd() == 3 and line.mdpt().ycrd() == 0)

    def test_mdpt(self):
        point = PointT(5,1)
        point2 = PointT(5,1)
        line = LineT(point, point2)
        self.assertTrue((line.mdpt()).xcrd() == 0 and line.mdpt().ycrd() == 0)

    def test_avg_function_in_LineADT(self):
        self.assertTrue(avg(4,2) == 3)

    def test_cen(self):
```

```
        circle = CircleT(PointT(5,5), 10)
        self.assertTrue((circle.cen()).xcrd() == 5 and (circle.cen()).ycrd() == 5)

    def test_rad(self):
        circle = CircleT(PointT(8,2), 25)
        self.assertTrue(circle.rad() == 25)

    def test_area(self):
        circle = CircleT(PointT(8,2), 25)
        self.assertTrue(circle.area() == pi* 25**2)

    def intersect(self):
        circle = CircleT(PointT(8,2), 25)
        circle2 = CircleT(PointT(100,100), 5)
        self.assertTrue( circle.intersect(circle2) == False)

    def test_connection(self):
        circle = CircleT(PointT(8,2), 25)
        circle2 = CircleT(PointT(10,10), 5)
        self.assertTrue(circle.connection(circle2).beg().xcrd() == 8 and
             circle.connection(circle2).beg().ycrd() == 2 and circle.connection(circle2).end().xcrd() ==
             10 and circle.connection(circle2).end().ycrd() == 10)


    def test_force(self):
        circle = CircleT(PointT(8,2), 25)
        circle2 = CircleT(PointT(10,10), 5)
        f = lambda f: f+1
        self.assertTrue(round(circle.force(f)(circle2)) == 1425882.0)

    def test_force_circle_with_force_is_very_small(self):
        circle = CircleT(PointT(1000,100000), 0.00001)
        circle2 = CircleT(PointT(10,10), 5)
        f = lambda f: f+1
        self.assertTrue(round(circle.force(f)(circle2)) == 0)

    def test_insideCircle(self):
        point = PointT(8,9)
        circle2 = CircleT(PointT(8,8), 5)
        self.assertTrue(insideCircle(point ,circle2) == True)

    def test_pushBack(self):
        circle2 = CircleT(PointT(10,10), 5)

        Deq.pushBack(circle2)
        self.assertTrue(Deq.back().rad() == circle2.rad())

    def test_pushBack_when_queue_is_full(self):
        Deq.popBack()


        c = CircleT(PointT(10,10), 5)
        for i in range(20):
            Deq.pushBack(c)
        self.assertRaises(Exception, Deq.pushBack, c)

    def test_pushFront(self):
        for i in range(21):
            Deq.popBack()


        c = CircleT(PointT(10,10), 5)
        c1 = CircleT(PointT(100,10), 5)

        Deq.pushFront(c)
        Deq.pushFront(c1)
        self.assertTrue(Deq.back().rad() == c.rad())

    def test_pushFront_when_queue_is_full(self):

        Deq.popBack()
        Deq.popBack()

        c = CircleT(PointT(10,10), 5)
        for i in range(21):
            Deq.pushBack(c)
        self.assertRaises(Exception, Deq.pushFront, c)

    def test_popBack(self):
```

```
        c = CircleT(PointT(10,10), 5)
        c1 = CircleT(PointT(100,10), 5)
        Deq.pushFront(c)
        Deq.pushFront(c1)
        self.assertTrue(Deq.popBack().rad() == c.rad())

    def test_popBack_when_the_queue_is_empty(self):
        Deq.popBack()
        Deq.popBack()
        self.assertRaises(Exception, Deq.popBack)

    def test_pop_Front(self):
        c = CircleT(PointT(10,10), 5)
        c1 = CircleT(PointT(100,10), 5)
        Deq.pushFront(c)
        Deq.pushFront(c1)

        self.assertTrue(Deq.popFront().rad() == c1.rad())

    def test_disjoint(self):
        c = CircleT(PointT(10,10), 5)
        Deq.pushFront(c)

        self.assertTrue(Deq.disjoint() == True)

    def test_sumFx(self):

        self.assertTrue(True)

if __name__ == '__main__':
    unittest.main()
```

F Makefile

```
all: clean refman.dvi

ps: refman.ps

pdf: refman.pdf

ps_2on1: refman_2on1.ps

pdf_2on1: refman_2on1.pdf

refman.ps: refman.dvi
	dvips -o refman.ps refman.dvi

refman.pdf: refman.ps
	ps2pdf refman.ps refman.pdf

refman.dvi: refman.tex doxygen.sty
	echo "Running latex..."
	latex refman.tex
	echo "Running makeindex..."
	makeindex refman.idx
	echo "Rerunning latex...."
	latex refman.tex
	latex_count=5 ; \
	while egrep -s 'Rerun (LaTeX|to get cross-references right)' refman.log && [ $$latex_count -gt 0
	    ] ;\
	do \
	  echo "Rerunning latex...." ;\
	  latex refman.tex ;\
	  latex_count=`expr $$latex_count - 1` ;\
	done

refman_2on1.ps: refman.ps
	psnup -2 refman.ps >refman_2on1.ps

refman_2on1.pdf: refman_2on1.ps
	ps2pdf refman_2on1.ps refman_2on1.pdf

clean:
	rm -f *.ps *.dvi *.aux *.toc *.idx *.ind *.ilg *.log *.out refman.pdf
```

G Partner's Code for CircleADT.py

```
#Michael Balas
#400023244
import pointADT
import lineADT
import math
## @file circleADT.py
#  @title CircleADT
#  @author Michael Balas
#  @date 2/2/2017
## @brief This class represents a circle ADT.
#  @details This class represents a circle ADT, with point cin (x, y)
#  defining the centre of the circle, and r defining its radius.
class CircleT(object):
        ## @brief Constructor for CircleT
        #  @details Constructor accepts one point and a number (radius)
        #  to construct a circle.
        #  @param cin is a point (the centre of the circle).
        #  @param rin is any real number (represents the radius of the circle).
        def __init__(self, cin, rin):
                self.c = cin
                self.r = rin
        ## @brief Returns the centre of the circle
        #  @return The point located at the centre of the circle
        def cen(self):
                return self.c
        ## @brief Returns the radius of the circle
        #  @return The radius of the circle
        def rad(self):
                return self.r

        ## @brief Calculates the area of the circle
        #  @return The area of the circle
        def area(self):
                return math.pi*(self.r)**2

        ## @brief Determines whether the circle intersects another circle
        #  @details This function treats circles as filled objects: circles completely
        #  inside other circles are considered as intersecting, even though
        #  their edges do not cross.  The set of points in each circle
        #  includes the boundary (closed sets).
        #  @param ci Circle to test intersection with
        #  @return Returns true if the circles intersect; false if not
        def intersect(self, ci):
                xDist = self.c.xc - ci.c.xcrd()
                yDist = self.c.yc - ci.c.ycrd()
                centerDist = math.sqrt(xDist ** 2 + yDist ** 2)
                rSum = self.r + ci.rad()
                return rSum >= centerDist
        ## @brief Creates a line between the centre of two circles
        #  @details This function constructs a line beginning at the centre of the
        #  first circle, and ending at the centre of the other circle.
        #  @param ci Circle to create connection with
        #  @return Returns a new LineT that connects the centre of both circles
        def connection(self, ci):
                return lineADT.LineT(self.c, ci.cen())
        ## @brief Determines the force between two circles given some parameterized
        #  gravitational law
        #  @details This functions calculates the force between two circles of unit
        #  thickness with a density of 1 (i.e. the mass is equal to the area). Any
        #  expression can be substituted for the gravitational law, f(r), or G/(r**2).
        #  @param f Function that parameterizes the gravitational law. Takes the distance
        #  between the centre of the circles and can apply expressions to it (e.g. multiply
        #  the universal gravitation constant, G, by the inverse of the squared distance between
        #  the circles).
        #  @return Returns the force between two circles
        def force(self, f):
                return lambda x: self.area() * x.area() * f(self.connection(x).len())
```

H Result of test cases

his was the result when my files were used to run the test module

- Ran 28 tests in 0.124 seconds
- However, for the last test method, nothing was implemented.

esults when partner's circleADT was used

- 27 tests passed
- 1 test failed
- One test failed because in partner's circleADT.py file, the insidecircle global function was not implemented.
- A picture of the output for running the partner's file has been shown in the next page.

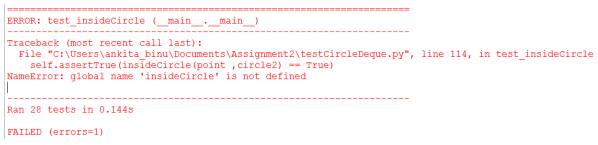

Figure 1: Result when partner's circleADT.py was used

I Discussion on Results

hen my files were used to run the testcircldeque module, the results were all correct. No errors were found when this was run

- In the rot() method, that was used in the pointADT module, round function was used. This was used so that, when sin and cos of values were calculated using math library, it will print a more precise answer.
- For example, when cos(pi) was implemented, python was givng a number really close to 1 but not a 1 exactly. The round function, will round the value up or down, depending on the value.
- This was necessary to do since these values are used by other methods in other classes. This will also make the method more reliable and correct.
- The validity of this method was verified by using 4 test cases for it that took 0 , positive and negative values.
- For the test cases that were returning objects, I tested them by checking their xcoord() and ycood().

J Issues with partner's file

- When my partner's circleADT was used for testing, only one error was found. Like I said before, this was becasue my partner did not implement the insidecircle global function in his circleADT module.

K Specifications of the modules

- I personally liked the mis format of specifications since they are less ambiguous than the informal format that was used.

- I learnt numerous things from doing this assignment. One of them was using the PYunit for test cases. Using this was so much more easier than typing the inpu and output for the last assignment. Pyunit will definetly help me to finish future projects in less time.

- The MIS specification for this assignment also enables all the students to have a similar design for this assignment. This was one of the reasons why there was less errors when the partner's file was used, unlike the previous assignment.

$$out := +(i : \mathbb{N} | i \in [0..|s| - 1] : s[i].\text{area}())$$

Figure 2: Specification for Deq_totalArea()

$$out := \frac{+(i : \mathbb{N} | i \in [0..|s| - 1] : s[i].\text{rad}())}{|s|}$$

Figure 3: Specification for Deq_averageRadius()

L Critique on Circle Module's interface

- For the most part, CircleADT module is pretty consistent. Some of the mehtod's naming were shortened. Methods like cen() and rad() should have been named center() and radius() to make it more consistent.
- This module is pretty essential since all the methods in this module is useful.
- This module is also general since a user can use this module for other purposes. However, this module is dependednt on lineADT and pointADT. If lineADT and pointADT were implemeted in CircleADT as classes, it would have been more general.
- Since no routines had 2 or more seperate functions /services to implement, this module is mininmal.
- This module was also opaque since if the user wants to change something, the user wouldn't have to change the whole module. Python does not support information hiding, but this can be partialy done by making certain methods private. This was not done since there was no purpose to do it in this module. The user can only change the values by calling the methods.

M Deq_disjoint() when there is one circle

– When there is one circle in the deque, the output of the mathematical expresion will be True. This is becasue when there is one circle in the deque, the deque's length will be 1. In the mathematical expression, 'i' will go from 0 to 0 and 'j' will go from 0 to 0. So the value of i and j will be 0. SInce the values of i and j are the same, it will unsatisfy the last condition. Therefore, the result will be true. This is the same result, my code will return when there is only one circle in the deque.