Assignment 1 Report

Tasnim Bari Noshin(noshint)

February 2, 2017

The purpose of this software design exercise is to write a Python program that creates, uses, and tests an ADT that stores circles. The program consist of the following files: `CircleADT.py`, `Statistics.py`, `testCircles.py` and `Makefile`, as shown in Appendices 1.1 to 1.5.

My partner's files:`CircleADT.py` and`Statistics.py`, is shown in Appendices 2.1 to 2.2.

A discussion comparing my test reults and my partner's test results is shown in Appendix 3.

1 Testing of the Original Program

1.1 Code for CircleADT.py

```
## @brief
# @details
import math
class CircleT:

    ## @brief __init__:initializes CircleT
    # @param x: x coordinate of the centre of the circle
    # @param y: y coordinate of the centre of the circle
    # @param r: the radius of the circle

    def __init__(self, x, y, r):
        self.x = float(x)
        self.y = float(y)
        self.r = float(r)
    ## @brief xcoord: a getter for x coordinate of the centre of the circle
    # @return x coordinate
    def xcoord(self):
        return self.x

    ## @brief ycoord: a getter for y coordinate of the centre of the circle
    # @return y coordinate
    def ycoord(self):
        return self.y

    ## @brief radius: a getter for the radius of the circle
    # @return radius of the circle
    def radius(self):
        return self.r

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

    ## @brief circumference: computes the circumference
    # @return circumference of the circle
    def circumference(self):
        return 2*math.pi*self.r

    ## @brief insideBox:checks to see if the circle is inside a given box
    # @param x0: x coordinate of the upper left corner of the box
    # @param y0: y coordinate of the upper left corner of the box
    # @param w:  the width of the box
    # @param h:  the height of the box
    # @return true if the circle is inside the box and otherwise false
    def insideBox(self, x0, y0, w, h):
        if (((self.x + self.r) <= (x0+w)) and (x0 <= (self.x - self.r)) and (((self.y + self.r) <= y0)
            and ((y0-h) <= (self.y - self.r)))):
            return True
        else:
            return False

    ## @brief intersect:checks to see if the new circle intersects with the given circle
    # @param c: a new circle
    # @return true if the new circle intersects with the given circle and otherwise false
    def intersect(self, c):
        if (((self.x - c.x)**2) + ((self.y - c.y)**2) <= ((c.r + self.r)**2)):
            return True
        else:
            return False

    ## @brief scale: changes radius of the given circle k times
    # @param k: a float number; changes the radius k times
    def scale(self, k):
        self.r = float(k) * self.r

    ## @brief translate: changes x and y coordinate of the given circle
    # @param dx: a float number; changes the centre of the circle by dx in the x direction
    # @param dy: a float number; changes the centre of the circle by dy in the y direction
    def translate(self, dx, dy):
        self.x = float(dx) + self.x
        self.y = float(dy) + self.y
```

1.2 Code for Statistics.py

```
## @file
# @details
import numpy
from CircleADT import *

## @brief average: computes the average radius of a given list of circles
# @param circles: list of circles
# @return the average radius
def average(circles):
    radii = []
    for i in range(len(circles)):
        radii += [circles[i].radius()]
    return numpy.average(radii)

## @brief stdDev: computes the standard deviation of a given list of circles
# @param circles: list of circles
# @return standard deviation
def stdDev(circles):
    radii = []
    for i in range(len(circles)):
        radii += [circles[i].radius()]
    return numpy.std(radii)

## @brief rank: computes a list ranked by radius from a given list of circles
# @param circles: list of circles
# @return a list ranked by radius
def rank(circles):
    radii = []
    for i in range(len(circles)):
        radii += [circles[i].radius()]

    index = list(range(len(radii)))
    index.sorted(key=lambda x: radii[x])
    ranks = [0] * len(index)
    for i, x in enumerate(index):
        ranks[x] = i+1
    ranks.reverse()
    return ranks
```

1.3 Code for testCircles.py

```
## @file
# @details
from CircleADT import *
from Statistics import *

## @brief main: calls testCirclT and testStatistics to run the test

def main():
    testCircleT()
    testStatistics()
## @brief testCircleT: tests all the functions in  CircleT() and prints whether they pass or fail.
def testCircleT():
    c = CircleT(0,0,4)
    d = CircleT(1,2,0)
    e = CircleT(10,10,2)
    f = CircleT(4,-1,3)
    g = CircleT(0,9,5)

    if(c.xcoord() == 0.0): print("Test1 for xcoord passed.")
    else: print("Test1 for xcoord failed.")
    if(d.xcoord() == 1.0): print("Test2 for xcoord passed.")
    else: print("Test2 for xcoord failed.")
    if(e.xcoord() == 10.0): print("Test3 for xcoord passed.")
    else: print("Test3 for xcoord failed.")
    if(f.xcoord() == 4.0): print("Test4 for xcoord passed.")
    else: print("Test4 for xcoord failed.")
    if(g.xcoord() == 0.0): print("Test5 for xcoord passed.")
    else: print("Test5 for xcoord failed.")

    if(c.ycoord() == 0.0): print("Test1 for ycoord passed.")
    else: print("Test1 for ycoord failed.")
    if(d.ycoord() == 2.0): print("Test2 for ycoord passed.")
    else: print("Test2 for ycoord failed.")
    if(e.ycoord() == 10.0): print("Test3 for ycoord passed.")
    else: print("Test3 for ycoord failed.")
    if(f.ycoord() == -1.0): print("Test4 for ycoord passed.")
    else: print("Test4 for ycoord failed.")
    if(g.ycoord() == 9.0): print("Test5 for ycoord passed.")
    else: print("Test5 for ycoord failed.")

    if(c.radius() == 4.0): print("Test1 for radius passed.")
    else: print("Test1 for radius failed.")
    if(d.radius() == 0.0): print("Test2 for radius passed.")
    else: print("Test2 for radius failed.")
    if(e.radius() == 2.0): print("Test3 for radius passed.")
    else: print("Test3 for radius failed.")
    if(f.radius() == 3.0): print("Test4 for radius passed.")
    else: print("Test4 for radius failed.")
    if(g.radius() == 5.0): print("Test5 for radius passed.")
    else: print("Test5 for radius failed.")

    if(c.area() == 50.26548245743669): print("Test1 for area passed.")
    else: print("Test1 for area failed.")
    if(d.area() == 0.0): print("Test2 for area passed.")
    else: print("Test2 for area failed.")
    if(e.area() == 12.566370614359172): print("Test3 for area passed.")
    else: print("Test3 for area failed.")
    if(f.area() == 28.274333882308138): print("Test4 for area passed.")
    else: print("Test4 for area failed.")
    if(g.area() == 78.53981633974483): print("Test5 for area passed.")
    else: print("Test5 for area failed.")

    if(c.circumference() == 25.132741228718345): print("Test1 for circumference passed.")
    else: print("Test1 for circumference failed.")
    if(d.circumference() == 0.0): print("Test2 for circumference passed.")
    else: print("Test2 for circumference failed.")
    if(e.circumference() == 12.566370614359172): print("Test3 for circumference passed.")
    else: print("Test3 for circumference failed.")
    if(f.circumference() == 18.84955592153876): print("Test4 for circumference passed.")
    else: print("Test4 for circumference failed.")
    if(g.circumference() == 31.41592653589793): print("Test5 for circumference passed.")
    else: print("Test5 for circumference failed.")

    if(c.insideBox(1,2,5,6) == True): print("Circle Test1 is inside the box")
    else: print("Circle Test1 is not inside the box")
    if(d.insideBox(1,2,5,6) == True): print("Circle Test2 is inside the box")
    else: print("Circle Test2 is not inside the box")
    if(e.insideBox(1,2,5,6) == True): print("Circle Test3 is inside the box")
    else: print("Circle Test3 is not inside the box")
    if(f.insideBox(1,2,6,6) == True): print("Circle Test4 is inside the box")
    else: print("Circle Test4 is not inside the box")
    if(g.insideBox(1,2,6,6) == True): print("Circle Test5 is inside the box")
    else: print("Circle Test5 is not inside the box")

    if(c.intersect(d) == True): print ("Circle Test1 intersects with Test2.")
    else:print ("Circle Test1 does not intersect with Test2.")
    if(d.intersect(e) == True): print ("Circle Test2 intersects with Test3.")
    else:print ("Circle Test2 does not intersect with Test3.")
    if(e.intersect(f) == True): print ("Circle Test3 intersects with Test4.")
    else:print ("Circle Test3 does not intersect with Test4.")
    if(f.intersect(g) == True): print ("Circle Test4 intersects with Test5.")
    else:print ("Circle Test4 does not intersect with Test5.")
    if(g.intersect(c) == True): print ("Circle Test5 intersects with Test1.")
    else:print ("Circle Test5 does not intersect with Test1.")

    c.scale(5)
    if(c.radius() == 20.0): print("Test1 for scale passed.")
    else:("Test1 for scale failed.")
    d.scale(5)
    if(d.radius() == 0.0): print("Test2 for scale passed.")
    else:("Test2 for scale failed.")
    e.scale(5)
    if(e.radius() == 10.0): print("Test3 for scale passed.")
    else:("Test3 for scale failed.")
    f.scale(5)
    if(f.radius() == 15.0): print("Test4 for scale passed.")
    else:("Test4 for scale failed.")
    g.scale(5)
    if(g.radius() == 45.0): print("Test5 for scale passed.")
    else:("Test5 for scale failed.")

    c.translate(2,3)
    if(c.xcoord() == 2.0 and c.ycoord() == 3.0): print("Test1 for translate passed.")
    else:("Test1 for translate failed.")
    d.translate(2,3)
    if(d.xcoord() == 3.0 and d.ycoord() == 5.0): print("Test2 for translate passed.")
    else:("Test2 for translate failed.")
    e.translate(2,3)
    if(e.xcoord() == 12.0 and e.ycoord() == 13.0): print("Test3 for translate passed.")
    else:("Test3 for translate failed.")
    f.translate(2,3)
    if(f.xcoord() == 6.0 and f.ycoord() == 2.0): print("Test4 for translate passed.")
    else:("Test4 for translate failed.")
    g.translate(2,3)
    if(g.xcoord() == 2.0 and g.ycoord() == 12.0): print("Test5 for translate passed.")
    else:("Test5 for translate failed.")

## @brief testStatistics: tests all the functions in  Statistics() and prints whether they pass or fail
def testStatistics():
    a = CircleT(0,0,4)
    b = CircleT(1,2,0)
    c = CircleT(10,10,2)
    d = CircleT(6,5,2)
    e = CircleT(0,9,5)
    circles=[a,b,c,d,e]

    if(average(circles) == 2.6000000000000001):print("Test for average passed.")
    else:print("Test for average failed.")

    if(stdDev(circles) == 1.7435595774162693):print("Test for stdDev passed.")
    else:print("Test for stdDev failed.")

    if(rank(circles) == [5, 3, 2, 1, 4]): print("Test for rank passed.")
    else:print("Test for rank failed.")


main()
```

1.4 test result for original code

```
Test1 for xcoord passed.
Test2 for xcoord passed.
Test3 for xcoord passed.
Test4 for xcoord passed.
Test5 for xcoord passed.
Test1 for ycoord passed.
Test2 for ycoord passed.
Test3 for ycoord passed.
Test4 for ycoord passed.
Test5 for ycoord passed.
Test1 for radius passed.
Test2 for radius passed.
Test3 for radius passed.
Test4 for radius passed.
Test5 for radius passed.
Test1 for area passed.
Test2 for area passed.
Test3 for area passed.
Test4 for area passed.
Test5 for area passed.
Test1 for circumference passed.
Test2 for circumference passed.
Test3 for circumference passed.
Test4 for circumference passed.
Test5 for circumference passed.
Circle Test1 is not inside the box
Circle Test2 is inside the box
Circle Test3 is not inside the box
Circle Test4 is inside the box
Circle Test5 is not inside the box
Circle Test1 intersects with Test2.
Circle Test2 does not intersect with Test3.
Circle Test3 does not intersect with Test4.
Circle Test4 does not intersect with Test5.
Circle Test5 intersects with Test1.
Test1 for scale passed.
Test2 for scale passed.
Test3 for scale passed.
Test4 for scale passed.
Test1 for translate passed.
Test2 for translate passed.
Test3 for translate passed.
Test4 for translate passed.
Test5 for translate passed.
Test for average passed.
Test for stdDev passed.
Test for rank passed.
```

1.5 Makefile

```
PY = python
PYFLAGS =
DOC = doxygen
DOCFLAGS =
DOCCONFIG = config
SRC = src/testCircles.py
.PHONY: all prog doc clean
test:
	python src/testCircles.py
doc:
	doxygen config
	cd latex && $(MAKE)
all: test doc
clean:
	rm -rf html
	rm -rf latex
```

2 Testing of Partner's Program

2.1 Code for Partner's CircleADT.py

```
## @file CircleADT.py
# @title CircleADT
# @author Adele Olejarz (400010177)
# @date Jan 28th, 2017
#MacID: olejarza

##want to use math.pi
import math


## @brief This class represents a circle
# @details This class represents a circle as a triple (x,y,r)
class CircleT(object):

    ##@brief Constructor for CircleT
    # @details Constructor accepts three real number parameters; x, y and r; where x and y define the
        origin of the circle, and r the radius.
    # @param x The x coordinate of the circle's origin
    # @param y The y coordinate of the circle's origin
    # @param r the radius of the circle
    # assume the user will not enter a 0 or negative radius
    def __init__(self,x,y,r):
        self.x = x
        self.y = y
        self.r = r

    ## @brief Returns the x coordinate of the circle's origin
    #  @return a real number for the x coordinate
    def xcoord(self):
        return self.x


    ## @brief Returns the y coordinate of the circle's origin
    #  @return a real number for the y coordinate
    def ycoord(self):
        return self.y

    ## @brief Returns the radius of the circle
    #  @return a real number for the radius
    def radius(self):
        return self.r

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

    ## @brief Returns the circumference of the circle
    #  @return the circumference of the circle
    def circumference(self):
        return 2*math.pi*self.r

    ## @brief Returns a boolean regarding wether or not the circle is contained in a box
    #  @details insideBox accepts 4 inputs, x,y,w,h where x is the x coordinate of the left side of a
        box, y is the y coordinate of the top of the box, w is the width of the box, and h is the
        height of the box. The function will return true if the circle is within the box, and false
        if it is not.
    #  @param x The x coordinate of the box's left side
    #  @param y The y coordinate of the box's top
    #  @param w the width of the box
    #  @param h the height of the box
    #  @return a boolean, which is true if the circle is within the box, false otherwise
    def insideBox(self,x,y,w,h):
        ##here i assume that the edges of the box count as "in the box" so that a circle with diameter
            equal to the side length of a box will fit in the box if and only if they share the same
            origin (centre point)
        if ((self.x + self.r) <= (x+w) and (self.x - self.r) >= x):
            if((self.y + self.r)<= y and (self.y - self.r) >= y - h):
              return True ##if the points on the circle farthest up down left and right are within the
                  box, the whole circle must be
            else: return False
        else: return False
```

```
## @brief Returns a boolean regarding whether or not the circle shares any points in common with
     an additional input circle, c.
#  @details If the combined radii are >= the distance, the circles MUST at least touch. By this
     logic, they intersect eachother as I consider the circumference a part of the circle
#  @param c The circle to which the calling circle will be compared.
#  @return a boolean, true if the circle shares any points with the input circle
def intersect(self,c):
    ## originDist is the distance between the two circles' origins. Will be compared to radii to
        determine intersection
    originDist = (math.sqrt((self.x - c.x)**2 + (self.y - c.y)**2))
    if ((c.r+self.r)>=originDist): return True
    else: return False

## @brief Multiplies the circle's radius by the input provided by the user
#  @details Takes the absolute value of the input. In theory a negative value would invert the
     radius, but since this is a circle, and the origin won't change, the effect would not be
     noticable. as such i take the absolute value of the scale.
#  @param k the float which to multiply the radius by
def scale(self,k):
    self.r = self.r *abs(k);

## @brief Translates the origin of the circle, dx in the x direction, and dy in the y direction
#  @param dx the amount to move the circle in the x direction
#  @param dy the amount to move the circle in the y direction
def translate(self,dx,dy):
    self.x += dx
    self.y += dy
```

2.2 Partner's Code: Statistics.py

```
## @file Statistics.py
#  @title Statistics
#  @author Adele Olejarz (400010177)
#  @date Jan 28th, 2017
#MacID: olejarza

import numpy as np
import CircleADT as circle

## @brief This method returns the average radius from a list of circles
#  @param circles a list of type CircleT
#  @return the average radius length in the list
def average(circles):
    ##list of radii
    radii = [circle.radius() for circle in circles]
    temp= np.average(radii)
    return temp

## @brief This method returns the standard deviation of radius from a list of circles
#  @param circles a list of type CircleT
#  @return the standard deviation of the radii ((average distance from average radius))
def stdDev(circles):
    ## list of radii
    stand = [circle.radius() for circle in circles]
    return np.std(stand)

## @brief This method takes a list of CircleT objects, and returns a list sorted by radius.
#  @param circles a list of type CircleT
#  @return A list of positions, based on rank of the corresponding circle's radius
def rank(circles):
    ##here I assume that a user will not input a negative value for a circle's radius, as this would
        be nonsensical
    ##a new list to store the rankings by radius
    ranklist=[]
    ##a list of just the radii, to be changed as we find the next highest radius
    radlist = [circle.radius() for circle in circles]
    ##initializing ranklist, so we can change the rank of every element as we find it.
    for x in range(0,len(radlist)): ranklist.append(0)

    ##the actual ranking algorithm
    for rank in range(1,len(radlist)+1):
        hi = 0        ##initialized to first element
        for i in range(0,len(radlist)):
            if (radlist[i] >=radlist[hi]):
                hi = i

        ranklist[hi] = rank
        radlist[hi] = -9991 ##this replaces it in the list, so we can find next highest element.


    return(ranklist)
```

2.3 test result for partner's code

```
Test1 for xcoord passed.
Test2 for xcoord passed.
Test3 for xcoord passed.
Test4 for xcoord passed.
Test5 for xcoord passed.
Test1 for ycoord passed.
Test2 for ycoord passed.
Test3 for ycoord passed.
Test4 for ycoord passed.
Test5 for ycoord passed.
Test1 for radius passed.
Test2 for radius passed.
Test3 for radius passed.
Test4 for radius passed.
Test5 for radius passed.
Test1 for area passed.
Test2 for area passed.
Test3 for area passed.
Test4 for area passed.
Test5 for area passed.
Test1 for circumference passed.
Test2 for circumference passed.
Test3 for circumference passed.
Test4 for circumference passed.
Test5 for circumference passed.
Circle Test1 is not inside the box
Circle Test2 is inside the box
Circle Test3 is not inside the box
Circle Test4 is inside the box
Circle Test5 is not inside the box
Circle Test1 intersects with Test2.
Circle Test2 does not intersect with Test3.
Circle Test3 does not intersect with Test4.
Circle Test4 does not intersect with Test5.
Circle Test5 intersects with Test1.
Test1 for scale passed.
Test2 for scale passed.
Test3 for scale passed.
Test4 for scale passed.
Test1 for translate passed.
Test2 for translate passed.
Test3 for translate passed.
Test4 for translate passed.
Test5 for translate passed.
Test for average passed.
Test for stdDev passed.
Test for rank failed.
```

3 Discussion

My partner's rank function fails for my tests. I create an anonymous function using sort(key = lambda value : value[0]) to sort my list of radii and reverse it as the initial result is from the biggest to the smallest value. My partner takes a different approach where she compares the radii with the zero index radius and changes the index if any of the radii is smaller than the first index.

I used math.pi from python's math library as I find it to be more accurate.