Assignment 1 Solution

Mohamed Bengezi bengezim

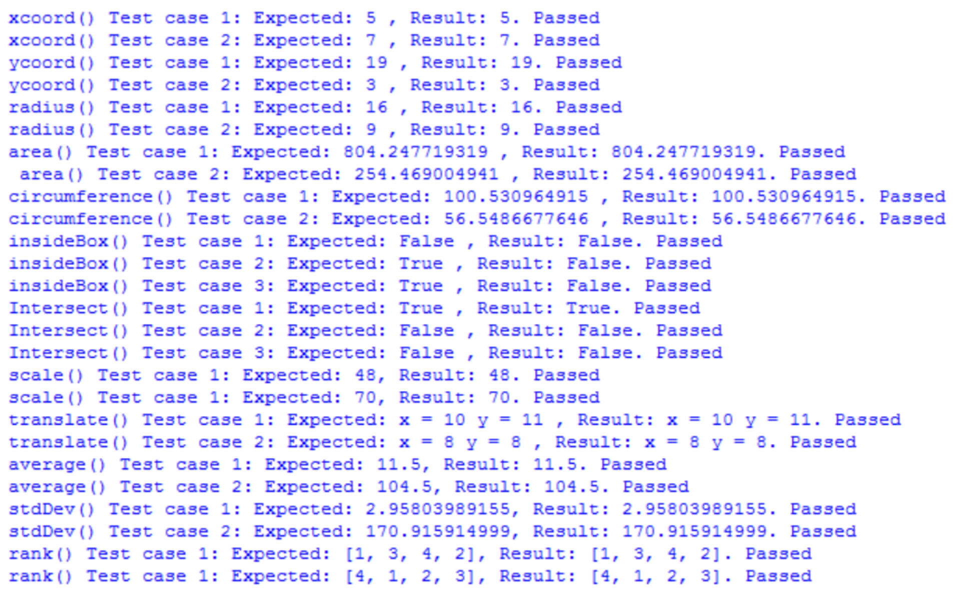
February 2, 2017

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 Appendix

Testing of the Original Program

The results of testing my files, specifically CircleADT.py and Statistics.py were all successful. Most of the simple functions had simple test cases, but more complicated ones, such as intersect(), or rank() had specialized ones. The following were specific test cases that were notable:

- In intersect(), I had the test case of two circles with the same centre, but different radii. This test case was expected to return True, seeing as they are fully intersecting. It successfully did so.
- Another intersect() test case that is notable is where the edges of the circles are just touching. With my assumptions, I expected the result to be false, seeing as they have not actually intersected one another. The actual result matched the expected
- In rank, I made the assumption that if there were two circles in the list with the same radius, the first one would be have a higher rank than the second one



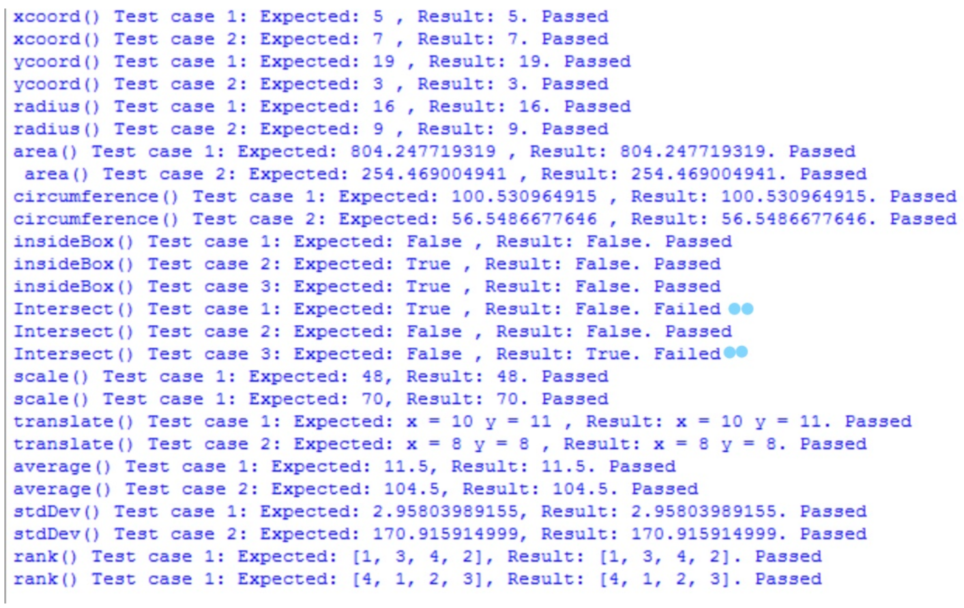
```
xcoord() Test case 1: Expected: 5 , Result: 5. Passed
xcoord() Test case 2: Expected: 7 , Result: 7. Passed
ycoord() Test case 1: Expected: 19 , Result: 19. Passed
ycoord() Test case 2: Expected: 3 , Result: 3. Passed
radius() Test case 1: Expected: 16 , Result: 16. Passed
radius() Test case 2: Expected: 9 , Result: 9. Passed
area() Test case 1: Expected: 804.247719319 , Result: 804.247719319. Passed
 area() Test case 2: Expected: 254.469004941 , Result: 254.469004941. Passed
circumference() Test case 1: Expected: 100.530964915 , Result: 100.530964915. Passed
circumference() Test case 2: Expected: 56.5486677646 , Result: 56.5486677646. Passed
insideBox() Test case 1: Expected: False , Result: False. Passed
insideBox() Test case 2: Expected: True , Result: False. Passed
insideBox() Test case 3: Expected: True , Result: False. Passed
Intersect() Test case 1: Expected: True , Result: True. Passed
Intersect() Test case 2: Expected: False , Result: False. Passed
Intersect() Test case 3: Expected: False , Result: False. Passed
scale() Test case 1: Expected: 48, Result: 48. Passed
scale() Test case 1: Expected: 70, Result: 70. Passed
translate() Test case 1: Expected: x = 10 y = 11 , Result: x = 10 y = 11. Passed
translate() Test case 2: Expected: x = 8 y = 8 , Result: x = 8 y = 8. Passed
average() Test case 1: Expected: 11.5, Result: 11.5. Passed
average() Test case 2: Expected: 104.5, Result: 104.5. Passed
stdDev() Test case 1: Expected: 2.95803989155, Result: 2.95803989155. Passed
stdDev() Test case 2: Expected: 170.915914999, Result: 170.915914999. Passed
rank() Test case 1: Expected: [1, 3, 4, 2], Result: [1, 3, 4, 2]. Passed
rank() Test case 1: Expected: [4, 1, 2, 3], Result: [4, 1, 2, 3]. Passed
```

Figure 1: Results of testing My files

Testing of the Partner Files

As seen in the next figure, my partner's files only failed the two test cases mentioned above. The main reason for this is that they have made different assumptions than me:

- One assumption is that when the circles are just touching, they are intersecting. This is a reasonable assumption.
- Another is that when the circles are fully overlapped, they are not intersecting. I believe this is not a reasonable assumption, seeing as mathematically they are intersecting in this case.



```
xcoord() Test case 1: Expected: 5 , Result: 5. Passed
xcoord() Test case 2: Expected: 7 , Result: 7. Passed
ycoord() Test case 1: Expected: 19 , Result: 19. Passed
ycoord() Test case 2: Expected: 3 , Result: 3. Passed
radius() Test case 1: Expected: 16 , Result: 16. Passed
radius() Test case 2: Expected: 9 , Result: 9. Passed
area() Test case 1: Expected: 804.247719319 , Result: 804.247719319. Passed
 area() Test case 2: Expected: 254.469004941 , Result: 254.469004941. Passed
circumference() Test case 1: Expected: 100.530964915 , Result: 100.530964915. Passed
circumference() Test case 2: Expected: 56.5486677646 , Result: 56.5486677646. Passed
insideBox() Test case 1: Expected: False , Result: False. Passed
insideBox() Test case 2: Expected: True , Result: False. Passed
insideBox() Test case 3: Expected: True , Result: False. Passed
Intersect() Test case 1: Expected: True , Result: False. Failed
Intersect() Test case 2: Expected: False , Result: False. Passed
Intersect() Test case 3: Expected: False , Result: True. Failed
scale() Test case 1: Expected: 48, Result: 48. Passed
scale() Test case 1: Expected: 70, Result: 70. Passed
translate() Test case 1: Expected: x = 10 y = 11 , Result: x = 10 y = 11. Passed
translate() Test case 2: Expected: x = 8 y = 8 , Result: x = 8 y = 8. Passed
average() Test case 1: Expected: 11.5, Result: 11.5. Passed
average() Test case 2: Expected: 104.5, Result: 104.5. Passed
stdDev() Test case 1: Expected: 2.95803989155, Result: 2.95803989155. Passed
stdDev() Test case 2: Expected: 170.915914999, Result: 170.915914999. Passed
rank() Test case 1: Expected: [1, 3, 4, 2], Result: [1, 3, 4, 2]. Passed
rank() Test case 1: Expected: [4, 1, 2, 3], Result: [4, 1, 2, 3]. Passed
```

Figure 2: Results of testing Partner files

Discussion

0.1 Issues with my code

As discussed earlier, the only controversial test cases were from intersect() and rank(). In summary, all my functions behaved expectedly. At first, I had tested my insideBox() method and found that it wasn't behaving expectedly. This was due to a wrong implementation of the if statement conditions. Once that was corrected, the functions all proceeded as expected

0.2 Issues with partners code

When it came to testing my partner's files, all functions behaved expectedly, except for inersect(). This was due to different assumptions made by me and my partner. This shows me that when working on projects with others, or even solely, you should take into consideration the assumptions that others might make, and how it could affect the correctness of your code.

0.3 Module Implementation and Specification

As for specification of the module's, the specification and implementation of the function were all very similar, with the exception of intersect(). This would explain why their intersect method failed my test cases. I directly used the distance between points formula and compared it with the radii, while they more so used thier own logic.

0.4 Handling π

I handled the value of π by using the math library in python. The reasoning for this is that seeing as it is a default python library, the constant is widely used, and accurate enough for this situation (11 decimal places). Seeing as it is not literally defined in any of my functions, it is defined explicitly, in the math library. The scope of π is then global, because it is used everywhere.

A Code for CircleADT.py

```
import math
## @brief CircleT is an ADT that stores data on a circle
#  @details It consists of various methods that return and/or manipulate the circles
class CircleT:
    ## @brief Initializes an object of type CircleT
    #  @param x is the x coordinate of the centre of the circle
    #  @param y is the y coordinate of the centre of the circle
    #  @param z is the radius of the circle
    def __init__(self,x,y,z):
        ## x-coordinate,y-coordinate,radius respectively
        self.x,self.y,self.z = x,y,z

    ## @brief Returns the x-coordinate of the centre of the circle
    #  @return The x-coordinate of the centre
    def xcoord(self):
        return self.x

    ## @brief Returns the y-coordinate of the centre of the circle
    #  @return The y-coordinate of the centre
    def ycoord(self):
        return self.y

    ## @brief Returns the radius of the circle
    #  @return The radius
    def radius(self):
        return self.z

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

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

    ## @brief Checks if the circle is inside the box
    #  @details Taking in the details of a box as input,
    #  it determines if the circle lies inside the box. Used pythagorean theorum
    #  @param x0 the x coordinate of the left side of a box
    #  @param y0 the y coordinate of the top of a box
    #  @param w the width of the box
    #  @param h the height of the box
    #  @return true if cirlce is inside the box
    ##check insideBox
    def insideBox(self,x0,y0,w,h):
        if (self.x-self.z) >= x0 and (self.x+self.z) <= (x0+w) and (self.y+self.z)<= y0+h and
            (self.y-self.z)>=(y0):
            return True
        else:
            return False

    ## @brief Checks if one circle intersects another
    #  @details Implemented using pythagorean theorum. Assumption made that two circles intersect only
        if the distance between centers is less than the two radii combined
    #  @param c the second circle to be compared
    #  @return true if cirlces intersect
    def intersect(self,c):
        ## r is xcoordinate of circle minus that of second circle
        r = self.x-c.x
        ## r is ycoordinate of circle minus that of second circle
        s = self.y-c.y
        ## dist is the result of the pythagorean theorum
        dist = math.sqrt(r**2 + s**2)
        if dist < (self.z + c.z):
            return True
        else:
            return False

    ## @brief Scales the radius of the circle
    #  @details radius is increased/decreased by factor k
    #  @param k the float k that the radius is scaled by
    def scale(self,k):
        self.z = self.z*k
```

```
## @brief Moves the centre of the circle
#  @details x and y coordinates are translated by factors dx, dy respectively
#  @param dx amount translated in the x
#  @param dy amount translated in the y
def translate(self,dx,dy):
    self.x = self.x + dx
    self.y = self.y + dy
```

B Code for Statistics.py

```
import numpy as np
import CircleADT
## @file Statistics.py contains methods for creating statistics for the CircleADT class
#  @brief Method uses include getting the average, standard deviation, and rank of the radii of Circles


## @brief Calculates the average radius of the given list
#  @details Takes in a list of CircleT objects as input, and outputs the average of all the radii
#  @param list1 is the list of CircleT objects
#  @return The average radius of the given list of CircleT objects
def average(list1):
    x = radiusList(list1)
    return np.average(x)

## @brief Calculates the standard deviation of the radii of all circles on list
#  @details Takes in a list of CircleT objects as input, and outputs the standard deviation of the
     radii
#  @param list1 is the list of CircleT objects
#  @return The standard deviation of the radii of the given list of CircleT objects
def stdDev(list1):
    x = radiusList(list1)
    return np.std(x)

## @brief Ranks the objects of the given list by radius
#  @details Takes in a list of CircleT objects as input, and outputs a list ranked by radius.
     Assumption made that if there is a tie, the first is labeled bigger then the second
#  @param list1 is the list of CircleT objects
#  @return The list of objects ranked by radius
def rank(list1):
    x = radiusList(list1)
    ranked = [0]*len(x)
    for i in range(len(x)):
        ranked[x.index(max(x))] = i + 1
        x[x.index(max(x))] = min(x) - 1

    return ranked

## @brief Creates a list of the radii for the list of CircleT object
#  @details Takes in a list of CircleT objects as input, and outputs a list of their radii
#  @param list1 is the list of CircleT objects
#  @return The list of radii
def radiusList(list1):
    rlist = []
    for i in list1:
        rlist.append(i.radius())
    return rlist
```

C Code for testCircles.py

```
import CircleADT as CADT
import Statistics as Stats
import math
import numpy as np

## @brief The Test class for CircleADT.py and Statistics.py
#  @details Consists of two test cases for each method
class TestCases():
    ## @brief Checks whether the method xcoord() works as expected
    #  @details Compares the actual result with the expected result
    #  @return A string displaying the result of both test cases
    def test_xcoord(self):
        #Test Case 1
        a = CADT.CircleT(5,5,5)
        if (a.xcoord() == 5):
            result = "xcoord() Test case 1: Expected: "+str(5)+" , Result: " + str(a.xcoord()) +".
                Passed \n"
        else:
            result = "xcoord() Test case 1: Expected: "+str(5)+" , Result: " + str(a.xcoord()) +".
                Failed \n"
        #Test Case 2
        b = CADT.CircleT(7,2,9)
        if (b.xcoord() == 7):
            result = result+ "xcoord() Test case 2: Expected: "+str(7)+" , Result: " + str(b.xcoord())
                +". Passed "
        else:
            result = result+ "xcoord() Test case 2: Expected: "+str(7)+" , Result: " + str(b.xcoord())
                +". Failed "
        return result

    ## @brief Checks whether the method ycoord() works as expected
    #  @details Compares the actual result with the expected result
    #  @return A string displaying the result of both test cases
    def test_ycoord(self):
        #Test Case 1
        a = CADT.CircleT(-3,19,5)
        if (a.ycoord() == 19):
            result = "ycoord() Test case 1: Expected: "+str(19)+" , Result: " + str(a.ycoord()) +".
                Passed \n"
        else:
            result = "ycoord() Test case 1: Expected: "+str(19)+" , Result: " + str(a.ycoord()) +".
                Failed \n"
        #Test Case 2
        b = CADT.CircleT(4,3,1)
        if (b.ycoord() == 3):
            result = result +"ycoord() Test case 2: Expected: "+str(3)+" , Result: " + str(b.ycoord())
                +". Passed "
        else:
            result = result +"ycoord() Test case 2: Expected: "+str(3)+" , Result: " + str(b.ycoord())
                +". Failed "
        return result

    ## @brief Checks whether the method radius() works as expected
    #  @details Compares the actual retrieved radius with the expected radius
    #  @return A string displaying the result of both test cases
    def test_radius(self):
        #Test Case 1
        a = CADT.CircleT(5,5,16)
        if (a.radius() == 16):
            result = "radius() Test case 1: Expected: "+str(16)+" , Result: " + str(a.radius()) +".
                Passed \n"
        else:
            result = "radius() Test case 1: Expected: "+str(16)+" , Result: " + str(a.radius()) +".
                Failed \n"
        #Test Case 2
        b = CADT.CircleT(7,2,9)
        if (b.radius() == 9):
            result = result +"radius() Test case 2: Expected: "+str(9)+" , Result: " + str(b.radius())
                +". Passed "
        else:
            result = result +"radius() Test case 2: Expected: "+str(9)+" , Result: " + str(b.radius())
                +". Failed "
        return result

    ## @brief Checks whether the method area() works as expected
    #  @details Compares the calculated area with the expected area
    #  @return A string displaying the result of both test cases
    def test_area(self):
        #Test Case 1
        a = CADT.CircleT(5,5,16)
        if (a.area() == math.pi*a.radius()**2):
            result = "area() Test case 1: Expected: "+str(math.pi*a.radius()**2)+" , Result: " +
                str(a.area()) +". Passed\n "
        else:
            result = "area() Test case 1: Expected: "+str(math.pi*a.radius()**2)+" , Result: " +
                str(a.area()) +". Failed\n "
        #Test Case 2
        b = CADT.CircleT(7,2,9)
        if (b.area() == math.pi*b.radius()**2):
            result = result +"area() Test case 2: Expected: "+str(math.pi*b.radius()**2)+" , Result: "
                + str(b.area()) +". Passed "
        else:
            result = result +"area() Test case 2: Expected: "+str(math.pi*b.radius()**2)+" , Result: "
                + str(b.area()) +". Failed "
        return result

    ## @brief Checks whether the method circumference() works as expected
    #  @details Compares the calculated circumference with the expected result
    #  @return A string displaying the result of both test cases
    def test_circumference(self):
        #Test Case 1
        a = CADT.CircleT(5,5,16)
        if (a.circumference() == 2*math.pi*a.radius()):
            result = "circumference() Test case 1: Expected: "+str(2*math.pi*a.radius())+" , Result: "
                + str(a.circumference()) +". Passed \n"
        else:
            result = "circumference() Test case 1: Expected: "+str(2*math.pi*a.radius())+" , Result: "
                + str(a.circumference()) +". Failed \n"
        #Test Case 2
        b = CADT.CircleT(7,2,9)
        if (b.circumference() == 2*math.pi*b.radius()):
            result = result + "circumference() Test case 2: Expected: "+str(2*math.pi*b.radius())+" ,
                Result: " + str(b.circumference()) +". Passed"
        else:
            result = result + "circumference() Test case 2: Expected: "+str(2*math.pi*b.radius())+" ,
                Result: " + str(b.circumference()) +". Failed"
        return result

    ## @brief Checks whether the method insideBox() works as expected
    #  @details Tested by determining the expected answer(T/F) by hand,
    #  then comparing with the method calculated answer.
    #  Assumption made that we are only looking at the fourth quadrant
    #  @return A string displaying the result of both test cases
    def test_insideBox(self):
        #Test Case 1
        a = CADT.CircleT(5,5,16)
        if (a.insideBox(5,5,5,5) == False):
            result = "insideBox() Test case 1: Expected: False , Result: " + str(a.insideBox(5,5,5,5))
                +". Passed \n"
        else:
            result = "insideBox() Test case 1: Expected: False , Result: " + str(a.insideBox(5,5,5,5))
                +". Failed \n"
        #Test Case 2
        b = CADT.CircleT(5,5,1)
        if (b.insideBox(1,1,10,10) == True):
            result = result + "insideBox() Test case 2: Expected: True , Result: " +
                str(a.insideBox(1,1,10,10)) +". Passed \n"
        else:
            result = result +"insideBox() Test case 2: Expected: True , Result: " +
                str(a.insideBox(1,1,10,10)) +". Failed \n"
        #Test Case 3
        b = CADT.CircleT(5,6,4)
        if (b.insideBox(0,0,10,12) == True):
            result = result +"insideBox() Test case 3: Expected: True , Result: " +
                str(a.insideBox(0,0,10,12)) +". Passed"
        else:
            result = result +"insideBox() Test case 3: Expected: True , Result: " +
                str(a.insideBox(0,0,10,12)) +". Failed"
        return result


    ## @brief Checks whether the method insideBox() works as expected
    #  @details Tested by determining the expected answer(T/F) by hand,
    #  then comparing with the method calculated answer
    #  @return A string displaying the result of both test cases
    def test_intersect(self):
        #Test Case 1
        a = CADT.CircleT(5,5,16)
        b = CADT.CircleT(5,5,10)
        if (a.intersect(b) == True):
            result = "Intersect() Test case 1: Expected: True , Result: " + str(a.intersect(b)) +".
                Passed \n"
        else:
            result = "Intersect() Test case 1: Expected: True , Result: " + str(a.intersect(b)) +".
                Failed \n"
        #Test Case 2
        a = CADT.CircleT(5,5,1)
        b = CADT.CircleT(1,1,1)
        if (a.intersect(b) == False):
            result = result + "Intersect() Test case 2: Expected: False , Result: " +
                str(a.intersect(b)) +". Passed \n"
        else:
            result = result + "Intersect() Test case 2: Expected: False , Result: " +
                str(a.intersect(b)) +". Failed \n"
        #Test Case 3
        a = CADT.CircleT(0,0,1)
        b = CADT.CircleT(2,0,1)
        if (a.intersect(b) == False):
            result = result + "Intersect() Test case 3: Expected: False , Result: " +
                str(a.intersect(b)) +". Passed"
        else:
            result = result + "Intersect() Test case 3: Expected: False , Result: " +
                str(a.intersect(b)) +". Failed"
        return result

    ## @brief Checks whether the method scale() works as expected
    #  @details Compares the new radius with the expected radius
    #  @return A string displaying the result of both test cases
    def test_scale(self):
        #Test Case 1
        a = CADT.CircleT(5,5,16)
        b = CADT.CircleT(5,5,10)
        r1 = a.radius()
        a.scale(3)
        if (a.radius() == r1*3):
            result = "scale() Test case 1: Expected: "+str(r1*3)+", Result: " + str(a.radius()) +".
                Passed\n"
        else:
            result = "scale() Test case 1: Expected: "+str(r1*3)+", Result: " + str(a.radius()) +".
                Failed\n"
        #Test Case 2
        r1 = b.radius()
        b.scale(7)
        if (b.radius() == r1*7):
            result = result + "scale() Test case 1: Expected: "+str(r1*7)+", Result: " +
                str(b.radius()) +". Passed"
        else:
            result = result + "scale() Test case 1: Expected: "+str(r1*7)+", Result: " +
                str(b.radius()) +". Failed"
        return result

    ## @brief Checks whether the method translate() works as expected
    #  @details Compares the new x and y coordinates with the expected ones
    #  @return A string displaying the result of both test cases
    def test_translate(self):
        #Test Case 1
        a = CADT.CircleT(5,5,16)
        b = CADT.CircleT(5,5,10)
        x1 = a.xcoord()
        y1 = a.ycoord()
        a.translate(5,6)
        if ((a.xcoord() == x1+5) and (a.ycoord() == y1+6)):
            result = "translate() Test case 1: Expected: x = "+str(x1+5)+" y = "+str(y1+6)+" , Result:
                x = " + str(a.xcoord()) +" y = "+str(a.ycoord())+". Passed\n"
        else:
            result = "translate() Test case 1: Expected: x = "+str(x1+5)+" y = "+str(y1+6)+" , Result:
                x = " + str(a.xcoord()) +" y = "+str(a.ycoord())+". Failed\n"
        #Test Case 2
        x2 = b.xcoord()
        y2 = b.ycoord()
        b.translate(3,3)
        if ((b.xcoord() == x2+3) and (b.ycoord() == y2+3)):
            result = result + "translate() Test case 2: Expected: x = "+str(x2+3)+" y = "+str(y2+3)+"
                , Result: x = " + str(b.xcoord()) +" y = "+str(b.ycoord())+". Passed"
        else:
            result = result + "translate() Test case 2: Expected: x = "+str(x2+3)+" y = "+str(y2+3)+"
                , Result: x = " + str(b.xcoord()) +" y = "+str(b.ycoord())+". Failed"
        return result
    ## @brief Checks whether the method average() works as expected
    #  @details Compares the calculated average with the pre-determined average
    #  @return A string displaying the result of both test cases
    def test_average(self):
        #Test Case 1
        a = CADT.CircleT(5,5,16)
        b = CADT.CircleT(5,5,10)
        c = CADT.CircleT(5,5,8)
        d = CADT.CircleT(1,1,12)
        x = [a,b,c,d]
        x1 = [a.radius(),b.radius(),c.radius(),d.radius()]
        if (Stats.average(x) == (np.average(x1))):
            result =  "average() Test case 1: Expected: "+str(np.average(x1))+", Result: " +
                str(Stats.average(x)) +". Passed\n"
        else:
            result =  "average() Test case 1: Expected: "+str(np.average(x1))+", Result: " +
                str(Stats.average(x)) +". Failed\n"
        #Test Case 2
        a = CADT.CircleT(5,5,-9)
        b = CADT.CircleT(5,5,20)
        c = CADT.CircleT(5,5,400)
        d = CADT.CircleT(1,1,7)
        x = [a,b,c,d]
        x1 = [a.radius(),b.radius(),c.radius(),d.radius()]
        if (Stats.average(x) == (np.average(x1))):
            result = result + "average() Test case 2: Expected: "+str(np.average(x1))+", Result: " +
                str(Stats.average(x)) +". Passed"
        else:
            result = result + "average() Test case 2: Expected: "+str(np.average(x1))+", Result: " +
                str(Stats.average(x)) +". Failed"

        return result

    ## @brief Checks whether the method stdDev() works as expected
    #  @details Compares the calculated standard deviation with the expected
    #  @return A string displaying the result of both test cases
    def test_stdDev(self):
        #Test Case 1
        a = CADT.CircleT(5,5,16)
        b = CADT.CircleT(5,5,10)
        c = CADT.CircleT(5,5,8)
        d = CADT.CircleT(1,1,12)
        x = [a,b,c,d]
        x1 = [a.radius(),b.radius(),c.radius(),d.radius()]
        if (Stats.stdDev(x) == (np.std(x1))):
            result =  "stdDev() Test case 1: Expected: "+str(np.std(x1))+", Result: " +
                str(Stats.stdDev(x)) +". Passed\n"
        else:
            result =  "stdDev() Test case 1: Expected: "+str(np.std(x1))+", Result: " +
                str(Stats.stdDev(x)) +". Failed\n"
        #Test Case 2
        a = CADT.CircleT(5,5,-9)
        b = CADT.CircleT(5,5,20)
        c = CADT.CircleT(5,5,400)
        d = CADT.CircleT(1,1,7)
        x = [a,b,c,d]
        x1 = [a.radius(),b.radius(),c.radius(),d.radius()]
        if (Stats.stdDev(x) == (np.std(x1))):
            result = result + "stdDev() Test case 2: Expected: "+str(np.std(x1))+", Result: " +
                str(Stats.stdDev(x)) +". Passed"
        else:
            result = result + "stdDev() Test case 2: Expected: "+str(np.std(x1))+", Result: " +
                str(Stats.stdDev(x)) +". Failed"
        return result
    ## @brief Checks whether the method rank() works as expected
    #  @details Compares the ranked list with the expected one
    #  @return A string displaying the result of both test cases
    def test_rank(self):
        #Test Case 1
        a = CADT.CircleT(5,5,16)
        b = CADT.CircleT(5,5,10)
        c = CADT.CircleT(5,5,8)
        d = CADT.CircleT(1,1,12)
        x = [a,b,c,d]
        x1 = [a.radius(),b.radius(),c.radius(),d.radius()]
        if (Stats.rank(x) == ([1,3,4,2])):
            result =  "rank() Test case 1: Expected: "+str([1,3,4,2])+", Result: " +
                str(Stats.rank(x)) +". Passed\n"
        else:
            result =  "rank() Test case 1: Expected: "+str([1,3,4,2])+", Result: " +
                str(Stats.rank(x)) +". Failed\n"
        #Test Case 2
        a = CADT.CircleT(5,5,-9)
        b = CADT.CircleT(5,5,20)
        c = CADT.CircleT(5,5,20)
        d = CADT.CircleT(1,1,7)
        x = [a,b,c,d]
        x1 = [a.radius(),b.radius(),c.radius(),d.radius()]
        if (Stats.rank(x) == ([4,1,2,3])):
            result = result + "rank() Test case 1: Expected: "+str([4,1,2,3])+", Result: " +
                str(Stats.rank(x)) +". Passed\n"
        else:
            result = result + "rank() Test case 1: Expected: "+str([4,1,2,3])+", Result: " +
                str(Stats.rank(x)) +". Failed\n"
        return result

b = TestCases()
print b.test_xcoord()
print b.test_ycoord()
print b.test_radius()
print b.test_area()
print b.test_circumference()
print b.test_insideBox()
print b.test_intersect()
print b.test_scale()
print b.test_translate()
print b.test_average()
print b.test_stdDev()
print b.test_rank()
```

D Partner's CircleADT.py

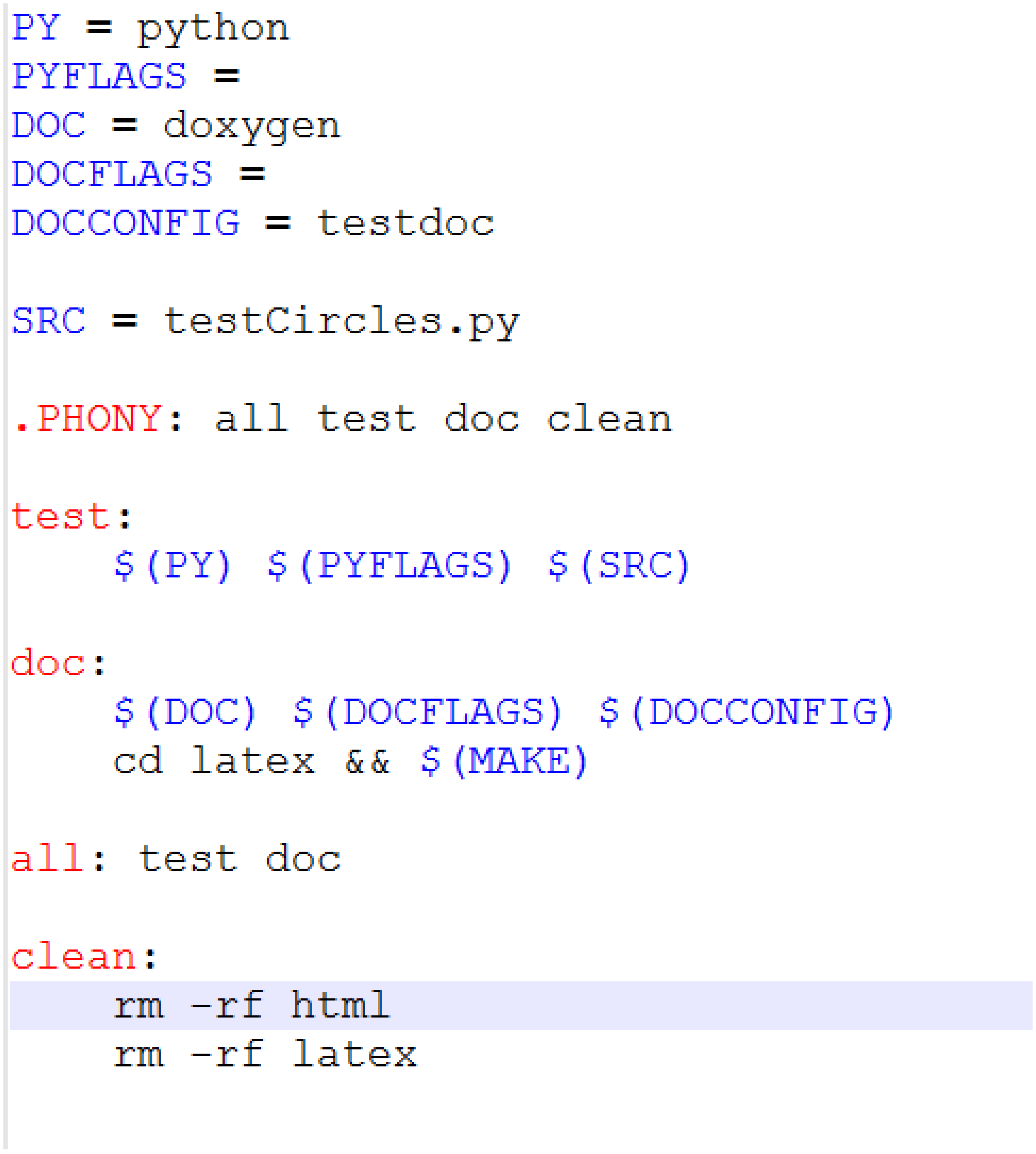
```
import math
from Statistics import average, stdDev, rank

class CircleT:
        ## @brief Initialize instance.
        def __init__(self, x, y, r):
                self.__x = x
                self.__y = y
                self.__r = r
        ## @brief Get xcoordinate.
        #  @return xcoordinate.
        def xcoord(self):
                return self.__x
        ## @brief Get ycoordinate.
        #  @return ycoordinate.
        def ycoord(self):
                return self.__y
        ## @brief Get radius.
        #  @return radius.
        def radius(self):
                return self.__r
        ## @brief Calculate area.
        #  @return area.
        def area(self):
                return (math.pi*self.__r**2)
        ## @brief Calculate circumference.
        #  @return circumference.
        def circumference(self):
                return(2*math.pi*self.__r)
    ## @brief Check if the circle is completely inside a box.
        #  @param x0 x coordinate
        #  @param y0 y coordinate
        #  @param w  width of box
        #  @param h  height of box
        #  @return (boolean) Is the circle completely inside the box?
        def insideBox(self, x0, y0, w, h):
                return (self.__x + self.__r) <= (x0 + w) and \
                           (self.__x - self.__r) >= x0       and \
                           (self.__y + self.__r) <= (y0 + h) and \
                           (self.__y - self.__r) >= y0
        ## @briefcheck if two circles intersect.
        #  @param data: a list of instances of CircleT.
        #  @return (boolean) Do the two circles intersect?
        def intersect(self, c):
                return ((self.__r-c.__r)**2 <= (self.__x-c.__x)**2+(self.__y-c.__y)**2 <=
                        (self.__r+c.__r)**2)
        ## @brief Scale radius
        #  @param k radius.
        def scale(self, k):
                self.__r = self.__r*k
        ## @brief translate circle.
        #  @param dx x translation
        #  @param dy y translation
        def translate(self, dx, dy):
                self.__x = self.__x + dx
                self.__y = self.__y + dy
```

E Partner's Statistics.py

```
import numpy
## @return the average of all circle radii
#  @param data a list of instances of CircleT
def average(data):
        rlist = []

        for x in data:
                rlist.append(x.radius())

        return numpy.average(rlist)

## @return the standard deviation of each circle in terms of radius size
#  @param data a list of instances of CircleT
def stdDev(data):
        mean = average(data)
        diffsSqrd = []

        for x in data:
                diffsSqrd.append((x.radius()-mean)**2)

        stdDev = numpy.average(diffsSqrd)
        return numpy.sqrt(stdDev)

## @return a list ranking each circle by radius size
#  @param data a list of instances of CircleT
def rank(data):
        alist = []
        blist = []
        clist = []
        index = 1
        index2 = 1
        #set up lists
        for n in data:
                alist.append(n.radius())
                blist.append(index)
                clist.append(index)
                index += 1
        #bubble sort the list of radii (alist), and do every.
        #change to the list of integers (blist) as well.
        for passnum in range(len(alist)-1,0,-1):
                for i in range(passnum):
                        if alist[i]<alist[i+1]:
                                temp = alist[i]
                                temp2 = blist[i]

                                alist[i] = alist[i+1]
                                blist[i] = blist[i+1]

                                alist[i+1] = temp
                                blist[i+1] = temp2

        for b in blist:
                clist[b-1] = index2
                index2 += 1

        return clist
```

F Makefile

Figure is on next page....



```
PY = python
PYFLAGS =
DOC = doxygen
DOCFLAGS =
DOCCONFIG = testdoc

SRC = testCircles.py

.PHONY: all test doc clean

test:
	$(PY) $(PYFLAGS) $(SRC)

doc:
	$(DOC) $(DOCFLAGS) $(DOCCONFIG)
	cd latex && $(MAKE)

all: test doc

clean:
	rm -rf html
	rm -rf latex
```

Figure 3: Makefile