2ME3 Assignment 1

Justin Staples (staplejw)

February 2, 2017

Contents

The purpose of this report is to document the results of creating and testing the following Python modules: `CircleADT.py`, `Statistics.py`, and `testCircles.py`. The report contains an analysis of the original test results, as well as the results from a secondary set of modules provided by a partner. All source files, as well as the corresponding makefile, can be found in the appendices (A - F).

1 Testing the Modules

The main objective for selecting the test cases was to choose tests that showcase the core functionality of each method. Basically, because only one test was required per method, it is essential to pick a test that shows the method is working in a common case. In the interest of time and simplicity, only the most basic tests were selected. However, for some methods, extra 'edge' cases were added (as specified by the interface).

For the purposes of testing, it was assumed that the person using the program would understand that the data type was intended to represent a circle, and would not input values other than real numbers (or non-negative real numbers for methods that involved a radius). So, no tests were selected that showcased this behaviour.

1.1 Original Modules

The main result of running the testing suite is that both modules (`CircleADT.py` and `Statistics.py`) passed all tests.

For the `area` method in `CircleADT.py`, an extra test was selected to see if the method would return an area equal to 0 in the case that the input radius was 0, which it did. A cirlce with a radius equal to 0 can be thought of as a point. The methods `insideBox` and `intersect` (from `CircleADT.py`) had a few extra cases added. This is because there are multiple common cases available to test. It was intended to show that the methods worked well in varying circumstances. For example, the `insideBox` method was tested three times: one test with the circle entirely outside the box, one with the circle entirely inside the box, and one with the circle completely inside the box, but intersecting the border of the box as well. A similar principle was used to test the `intersect` method: one test for two circles that have 0 points in common, one with two circles that have 1 point in common, and another with two circles that have an infinite number of points in common (that is to say they overlap). Lastly, for the `rank` method, a small group of circles were chosen, of varying radii. The first test follows the example from the MIS exactly, sorting an array of values where each element is different. The second test ranks an array of values where there is a tie for first place. The last test ranks an array of values where there is a tie for another place that is not first.

It might be worth noting that the tests were planned and created after the modules were written, and that the modules were not edited during the implementation of the testing suite.

1.2 Partners Modules

For the partners code, the main result of running the testing suite is that the `CircleADT.py` module passed all tests, while the `Statistics.py` did not. The reason that this happened is because of a design decision that differed between the partners code and the original implementation. The original source code will allow for the `rank` method in `Statistics.py` to skip values after a tie (e.g. it might give a ranking like [1, 2, 2, 4]), while the partners implementation does not, and ensures that ranks always step to the next available integer (e.g. it would give a ranking like [1, 2, 2, 3]). The details of how to handle ties were not made perfectly clear in the specification, and so it not hard to imagine why there were differences in the two implementations.

Other than the `rank` method, all the other methods from the partners modules gave the expected output and passed the tests.

2 Discussion

This section outlines some of the shortcomings of the original program code, the partners implementation, as well as the specification of the modules given in the assignment. Some possible improvements are mentioned, along with a brief discussion of the lessons learned during this exercise.

2.1 Original Modules

In a real world application, it might not be best to assume that users will always enter valid input types. Entering other types (boolean, string, etc.) into the methods as arguments would certainly produce an error and cause the program to crash. Although handling these types of errors was not specified in the MIS, it might be a good idea to try to handle them, as it allows for a more robust program.

Although performance was not specified in the MIS as something of critical importance, it might have been worth it to consider different implementations of certain methods. The implementations that were chosen were meant to be 'quick and dirty', and this could be considered a fault if someone using the module was intersted in speed or performance. For example, the algorithm used for ranking was more of a brute force method, and more optimal algorithms might be considered depending on the users needs.

It was not specified in the MIS though, so the module remains correct even with a slower implementation.

In terms of documentation, the Doxygen comments in the modules, in general, give a complete outline of the methods and their parameters. However, it was left out in the documentation of the `average` and `stdDev` methods, that the `numpy` library is used to do the calculations. This might have been important to know for someone using the module or reading the documentation, and was not intended to be left out. It was intended to ensure comments always had a hard wrap of 80 character width, but there were a few instances where this is violated. This is a mistake that could be fixed in future with more careful planning and attention to detail.

Lastly, there was a strong effort made to make sure that the style of the code was consistent (indentation, spacing, etc.), but there are a few spots where there is slight deviation. This is not considered a major problem, but something that could be improved upon for future projects.

Overall, it is suggested by the results of the tests that the original modules are correct. However, a more rigorous and systematic testing suite could have been designed to further establish the programs correctness.

2.2 Partners Modules

The implementations written by the partner were overall quite similar to that of the original. The details of the code and overall style are quite similar, which makes perfect sense considering the nature of the assignment. This means that both sets of modules contain similar errors. The partners code also has slight inconsistencies in formatting (this is a very small issue, but could help readability if improved), but overall meets the specifications in terms of functionality.

It is worth noting that the partners implementation of the `CircleADT.py` module contains an extra method called `circumference`, which simply returns the circumference of a circle. This method was not part of the specification and should be viewed as unnecessary or incorrect to include it.

2.3 Usage of External Libraries

An important design decision to be made during the implementation of this MIS is how to define the constant π. There are a few available options that were considered. The first, and perhaps simplest way to approximate π would simply be to truncate the decimal at an arbitrary point, leaving only, say, the first 10 values after the decimal. This would lead to a very close approximation for calculating area or other things that required π. Another similar way to to estiamte π would be to use its definition as an infinte sum, and

instead of choosing an arbitrary number of decimals, you could just choose an arbitrary number of terms to include in the sum as an approximation. Although both of these methods could prove to be quite accurate, the problem with them is that the cutoffs for their precision are arbitrary and will depend on the person implementing the method. If many different approximations for π were used by many different implementations, then surely testing the `area` method with different code would lead to failures because all the results would be very close but not exactly the same. The way to combat this is to choose a standard value for π, that is both accurate and readily available to all programmers. For these reasons, the value of π that was selected for the original implementations (and also selected by the partner) was the value given by the Python standard math library. Standardization is desirable to ensure that the code can be used again and again by many different clients without causing confusion, thereby increasing reusability.

Another part of the specification for this assignment was to use the `numpy` library for calculating averages and standard deviations. Using an external library like this has many benefits. It saves the programmer time and space. Others have already developed methods for calcuating average and standard deviation. Using the library helps increase productivity because the programmer does not have to implement these functions themselves. These functions are also essentialy guaranteed to be fast and correct. Although average and standard deviation are simple to implement, there may be much more complex functions, such as fast fourier transforms, or inverting a large matrix, where it is best to use the tools found in a standard library. This is because these libraries have already been tested extensively, meaning that they are correct and reliable.

2.4 Closing Remarks

It would have been interesting to see how the original modules fared in a more rigorous testing environment. There are many other ways of testing the modules that were not explored. For example, as mentioned above, the modules might have been tested to see how they handle different input types. Perhaps even more complicated tests could have been designed to test how the modules reacted to a series of method calls, all in a row. For example, start with an instance of `CircleT`, then scale it, translate it, and then test to see if it intersects with a particular box. Testing the methods in different combinations might reveal more about potential faults in the code.

It is certainly worth discussing the nature of the assignment, in terms of the specification and how it left some decisions up to the programmer. The choice of which value for π to use, or which definition of intersection to use, and also how to rank a set of values which includes ties, all reveal that even slight ambiguities in the specification will lead to many different implementations. This could cause many problems in a real world application, where the user is expecting one thing and gets another because it was not

made clear to the progammer exactly what to do. Surely, to avoid this kind of confusion, program specifications should be made as formal as possible, to allow the mapping from MIS to programming language more automatic.

A Original Code for CircleADT.py

```
## @file CircleADT.py
#  @title CircleADT
#  @author Justin Staples

import math

## @brief This class represents a circle.
#  @details This class represents a circle as abstract data type.
#  The circle is characterized with a center point with x and y
#  coordinates (x, y), and a radius, r.
class CircleT:
    ## @brief This is the constructor for CircleT
    #  @details The constructor takes in three parameters. Two are
    #  for the center of the circle (x and y) and a third which
    #  gives the radius.
    #  @param x x-coordinate of the center of the circle.
    #  @param y y-coordinate of the center of the circle.
    #  @param r the radius of the circle.
    def __init__(self, x, y, r):
        self.x = x
        self.y = y
        self.r = r

    ## @brief Getter method for x-coordinate.
    #  @return Returns the x-coordinate of an intance of CircleT.
    def xcoord(self):
        return self.x

    ## @brief Getter method for y-coordinate.
    #  @return Returns the y-coordinate of an intance of CircleT.
    def ycoord(self):
        return self.y

    ## @brief Getter method for the radius.
    #  @return Returns the radius of an intance of CircleT.
    def radius(self):
        return self.r

    ## @brief Method for calculating the area of an instance of CircleT.
    #  return Returns the area of an instace of CircleT.
    def area(self):
        return math.pi * self.r ** 2

    ## @brief Checks to see if an instance of CircleT is in
    #  side of a rectangular box.
    #  @details This method will calculate a range of values that the
    #  center of the circle is allowed to be in, given it's radius. Points
    #  of the circle that lie on the edge of the box are OK.
    #  @param x0 X-coordinate of the left side of the box.
    #  @param y0 Y-coordinate of the top side of the box.
    #  @param w Width of the box.
    #  @param h Height of the box.
    #  @return The function returns a boolean value. True, if the center
    #  of the circle is within the correct range, and False otherwise.
    def insideBox(self, x0, y0, w, h):
        upper_x = x0 + w - self.r
        lower_x = x0 + self.r
        upper_y = y0 - self.r
        lower_y = y0 - h + self.r
        if (lower_x <= self.x <= upper_x) and (lower_y <= self.y <= upper_y):
            return True
        else:
            return False

    ## @brief Checks to see if two instances of CircleT intersect eachother.
    #  @details The function will do a check to see how far away the two
    #  circles center's are away from eachother. This will get compared to the
    #  sum of the radii to see if the two circles share any points in common.
    #  @param c Another instance of CircleT to test with.
    #  @return The function returns a boolean value. True, if the circles share
    #  any common points (even just one), and False otherwise. That is to say,
    #  I have made the assumption that intersection is satisfied with just one
    #  point, and the method uses the <= operator to allow this.
    def intersect(self, c):
        dist = math.sqrt((self.x - c.x) ** 2 + (self.y - c.y) ** 2)
        if (dist <= self.r + c.r):
```

```
            return True
        else:
            return False

    ## @brief Scales the radius of an instance of CircleT by a factor, k.
    #  @param k The scale factor.
    def scale(self, k):
        self.r = k * self.r

    ## @brief Translates an instance of CircleT .
    #  @details The circle will be translated 'dx' units along the x-axis
    #  and 'dy' units along the y-axis.
    #  @param dx Determines how much the circle will be translated along the
    #  x-axis.
    #  @param dy Determines how much the circle will be translated along the
    #  y-axis.
    def translate(self, dx, dy):
        self.x = self.x + dx
        self.y = self.y + dy
```

B Original Code for Statistics.py

```
## @file Statistics.py
#  @title Statistics
#  @author Justin Staples
#  @details This file contains three functions for analyzing
#  the radii of a list of circles. It is implemented as a library.

import numpy as np

## @brief This method calculates the average radius, given a list of
#  circles.
#  @param list This is the input list of instances of CircleT.
#  @return The value that is returned is average radius of all the
#  circles in the list.
def average(list):
        x = np.zeros(len(list))
        for i in range(0, len(list)):
                x[i] = list[i].r
        return np.average(x)

## @brief This method calculates the standard deviation of the radii,
#  given a list of circles.
#  @param list This is input list of instances of CircleT.
#  @return The value that is returned is the standard deviation of
#  the radii of all circles in the list.
def stdDev(list):
        x = np.zeros(len(list))
        for i in range(0, len(list)):
                x[i] = list[i].r
        return np.std(x)

## @brief This method ranks a list of circles according to their radius.
#  @details This method first uses the input list to create a new list
#  with just the radii. Then, it iterates through this new list, and for
#  each element, checks how many elements in the list are larger than it.
#  The counter variable is initialized to 1. This is because I have assumed
#  that the lowest possible rank value is 1. I have also assumed that circles
#  with the same radii will receive the exact same rank, one below the next largest radius. After
    ranking
#  elements that are part of a tie, the rank value will go back to normal (will
#  jump values by at least 2 depending on how many elements are involved in the tie).
#  This is evident in my test cases for this method.
#  @param list This is the input list of instances of CircleT.
#  @return The method return a list of integers with the same length as
#  the input list. Each element of the list gives a rank which corresponds
#  to the instance of CircleT with the same index.
def rank(list):
        x = []
        a = []
        for i in range(0, len(list)):
                x.append(list[i].r)
        for i in range(0, len(list)):
                rank = 1
                for j in range(0, len(list)):
                        if (x[j] > x[i]):
                                rank += 1
                a.append(rank)
        return a
```

C Code for testCircles.py

```
## @file testCircles.py
#  @title Tests
#  @author Justin Staples
#  @details This file contains a series of tests for all the methods
#  in CircleADT.py and Statistics.py modules. Each method has at least
#  one test case. For the most part, I decided to pick basic test cases
#  which can outline that the core function of each method was implemented
#  properly. For certain methods, I picked a few extra cases which test
#  'interesting' behaviour such as calculating the area of a circle with
#  radius = 0, or ranking lists of circles... When this module is
#  interpreted, all of the tests run automatically, and each one is asserted
#  True or False, based on the expected values. In all cases, the desired
#  assertion is True (I realize that some outputs for boolean values are
#  expected to be False, but this module checks that False == False, and
#  will print True). All of the tests print to the terminal and show the
#  input, expected output and result. For the tests that I have chosen, all
#  assert to True, which gives at least some indication that the other
#  modules were implemented properly. I will give a more detailed explanation
#  all the tests in my report.

from CircleADT import *
from Statistics import *

print "--------------------------------"
test = [CircleT(1, 2, 3), CircleT(-1.7, 2.3, 0)]
print "testing constructor/accessors:"
print "TEST: CircleT(1, 2, 3), EXPECTED: x = 1, y = 2, r = 3"
print test[0].xcoord() == 1, test[0].ycoord() == 2, test[0].radius() == 3
print "TEST: CircleT(-1.7, 2.3, 0), EXPECTED: x = -1.7, y = 2.3, r = 0"
print test[1].xcoord() == -1.7, test[1].ycoord() == 2.3, test[1].radius() == 0
print "--------------------------------"

test = [CircleT(3, 4, 0), CircleT(2, 8, 10)]
print "testing area:"
print "TEST: CircleT(3, 4, 0), EXPECTED: area = 0"
print test[0].area() == 0
print "TEST: CircleT(2, 8, 10), EXPECTED: area = (math.pi)*(10**2) = 314.159..."
print test[1].area() == (math.pi)*(10**2)
print "--------------------------------"

test = CircleT(0, 0, 1)
print "testing insideBox:"
print "TEST: CircleT(0, 0, 1), box parameters given by [-5, 5, 1, 1] EXPECTED: False"
print test.insideBox(-5, 5, 1, 1) == False
print "TEST: CircleT(0, 0, 1), box paramaters given by [-2, 2, 4, 4] EXPECTED: True"
print test.insideBox(-2, 2, 4, 4) == True
print "TEST: CircleT(0, 0, 1), box paramaters given by [-1, 1, 2, 2] EXPECTED: True"
print test.insideBox(-1, 1, 2, 2) == True
print "--------------------------------"

test = [CircleT(0, 0, 1), CircleT(-1, 0, 1), CircleT(2, 0, 1)]
print "testing intersect:"
print "TEST: CircleT(0, 0, 1) and CircleT(1, 0, 1), EXPECTED: True"
print test[0].intersect(test[1]) == True
print "TEST: CircleT(0, 0, 1) and CircleT(2, 0, 1), EXPECTED: True"
print test[0].intersect(test[2]) == True
print "TEST: CircleT(-1, 0, 1) and CircleT(2, 0, 1), EXPECTED: False"
print test[1].intersect(test[2]) == False
print "--------------------------------"

test = CircleT(2, -1, 6)
test.scale(3)
print "testing scale:"
print "TEST: CircleT(2, -1, 6), k = 3, EXPECTED: r = 6*3 = 18"
print test.r == 18
print "--------------------------------"

test = CircleT(1, 6, 11)
test.translate(3, -4)
print "testing translate:"
print "TEST: CircleT(1, 6, 11), dx = 3, dy = -4, EXPECTED: x = 4, y = 2"
print test.x == 4, test.y == 2
print "--------------------------------"

test = [CircleT(-1, -1, 3), CircleT(4, 4, 4), CircleT(7, 6, 5)]
print "testing average:"
```

```
print "TEST: circles with radii of 3, 4, 5, EXPECTED: average = 4"
print average(test) == 4
print "------------------------------"

print "testing stdDev:"
print "TEST: circles with radii of 3, 4, 5, EXPECTED: stdDev = 0.8165..."
print stdDev(test) == math.sqrt((((3.0-4)**2 + (4-4)**2 + (5-4)**2)/3))
print "------------------------------"

test_1 = [CircleT(1, 2, 6), CircleT(1, 2, 5), CircleT(1, 2, 11), CircleT(1, 2, 9)]
test_2 = [CircleT(1, 2, 6), CircleT(1, 2, 5), CircleT(1, 2, 11), CircleT(1, 2, 11)]
test_3 = [CircleT(1, 2, 9), CircleT(1, 2, 5), CircleT(1, 2, 11), CircleT(1, 2, 9)]
print "testing rank:"
print "TEST: circles with radii of [6, 5, 11, 9], EXPECTED: [3, 4, 1, 2]"
print rank(test_1) == [3, 4, 1, 2]
print "TEST: circles with radii of [6, 5, 11, 11], EXPECTED: [3, 4, 1, 1]"
print rank(test_2) == [3, 4, 1, 1]
print "TEST: circles with radii of [9, 5, 11, 9], EXPECTED: [2, 4, 1, 2]"
print rank(test_3) == [2, 4, 1, 2]
```

D Code for Makefile

```
PY = python
PYFLAGS =
DOC = doxygen
DOCFLAGS =
DOCCONFIG = staplejw_A1

SRC = src/testCircles.py

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

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

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

E Partners Code for CircleADT.py

```
## @file
# @brief
# @author Umme Salma Gadriwala
# @date 28-Jan-2017
#CircleADT.py

import math
## @brief CircleT - abstract data type
# @details: contains xcoord, ycoord, radius, area, circumference, insideBox, intersect, scale,
    translate methods. \n
# Assumption: A point is a CircleT with radius 0.
class CircleT(object):

    ## @brief CircleT constructor - creates an instance of CircleT
    # @param x: x-coordinate of center of circle, type: real
    # @param y: y-coordinate of center of circle, type: real
    # @param r: radius of circle, type: real
    # @return instance of CircleT
    def __init__(self, x, y, r):
        self.x = x
        self.y = y
        self.r = r

    ## @brief Gets x-coordinate of center of circle
    # @ return float
    def xcoord(self):
        return self.x

    ## @brief Gets y-coordinate of center of circle
    # @ return float
    def ycoord(self):
        return self.y

    ## @brief Gets radius of circle
    # @ return float
    def radius(self):
        return self.r

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

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

    ## @brief Returns true if circle inside box, false otherwise
    # @details The function assumes that a circle that touches the egdes of the box is inside it.
    # @param x0 : x coordinate of the left side of a box
    # @param y0 :  y coordinate of the top of a box
    # @param w :  width of the box
    # @param h :   height of the box
    # @return boolean
    def insideBox(self, x0, y0, w, h):
        x = self.x
        y = self.y
        r = self.r
        return (x+r) <= (x0 + w)  \
               and (x-r) >= x0          \
               and (y+r) <= y0          \
               and (y-r) >= (y0 - h)

    ## @brief Returns true if circles intersect
    # @details The function returns true if the points inside the two circles (including those on the
        circumference) overlap.
    # @param c: object of type CircleT
    # @return boolean
    def intersect(self, c):
        distance = math.sqrt ((self.x - c.x)**2 + (self.y - c.y)**2)
        if distance > (self.r + c.r): return False
        else: return True

    ## @brief Mutates radius of circle by factor of k
    # @param k: float, scales radius by k
```

```
def scale(self, k):
    self.r = self.r*k

## @brief Mutates center of circle by dx, dy
# @param dx: float, change in x-xoordinate
# @param dy: float, change in y-coordinate
def translate(self, dx, dy):
    self.x = self.x + dx
    self.y = self.y + dy
```

F Partners Code for Statistics.py

```
## @file
# @brief
# @author Umme Salma Gadriwala
# @date 28-Jan-2017

from CircleADT import *
import numpy

## @brief returns the average radius of a list of CircleT
# @details Uses numpy library to calculate average radius. Assumes listCircleT is a list containing
    objects of type CircleT only.
# @param listCircleT: type: list of instances of CircleT
# @return float
def average(listCircleT):
    # listCircleT: type: list of instances of CircleT
    # return type: float
    # returns average radius of all circles in list
    radii = []
    for circle in listCircleT:
        radii += [circle.radius()]
    return numpy.average(radii)

## @brief returns the population standard deviaton of radii of a list of CircleT
# @details Uses numpy library to calculate standard deviation. Assumes listCircleT is a list
    containing objects of type CircleT only.
# @param listCircleT: type: list of instances of CircleT
# @return float
def stdDev(listCircleT):
    # listCircleT: type: list of instances of CircleT
    # return type: float
    # returns standard deviation of radii of all circles in list
    radii = []
    for circle in listCircleT:
        radii += [circle.radius()]
    return numpy.std(radii)

## @brief returns a ranking list by radii
# @details Assumes listCircleT is a list containing objects of type CircleT only. For two CircleT of
    same length, they have the same rank.
# @param listCircleT: type: list of instances of CircleT
# @return list of int
def rank(listCircleT):
    # listCircleT: type: list of instances of CircleT
    # return type: list of CircleT
    # returns a ranking list by radii
    indexRadius = []
    index = 0
    for circle in listCircleT:
        indexRadius += [(index, circle.radius())]
        index += 1
    indexRadius.sort(key=lambda tup: tup[1])
    rank = [indexRadius[0][0] + 1]
    radius = indexRadius[0][1]
    for i in indexRadius[1:]:
        if i[1] == radius: rank = [rank[0]] + rank
        else:
            rank = [i[0] + 1] + rank
            radius = i[1]
    return rank
```