Assignment 1 Solution

SFWR ENG 2AA4

February 1, 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 A to C.

Roadmap of the report.

Testing of the Original Program

Body of the report, including additional sections.

...

A Code for CircleADT.py

```
## @file Box3D.py
#  @title Box3D
#  @author Mustafa Haddara, Steven Palmer
#  @date 1/8/2017

## @brief This class represents a rectangle.
#  @details This class represents a rectangle as (x,y,z) coordinate representing the
#  front-facing top-left corner, and a (w,h,d) tuple representing the width, height and depth.
class Box3D(object):

    ## enum value FRONT
    FRONT = 0
    ## enum value TOP
    TOP = 1
    ## enum value SIDE
    SIDE = 2


    ## @brief Constructor for Box3D
    #  @details Constructor accepts two parameters for the top left corner and size dimensions.
    #  @param corner (x,y,z) tuple for the top left corner.
    #  @param size (w,h,d) tuple containing the width, height and depth.
    def __init__(self, corner, size):
        if len(corner) != 3 or len(size) != 3:
            raise ValueError(”Invalid argument size”)
        # double underscore (__) before the identifier is python convention
        # for private variables
        self.__frontTopLeftCorner = corner
        self.__dimensions = size


    ## @brief Returns the front top left corner tuple.
    #  @return (x,y,z) tuple for the top left corner.
    def getFrontTopLeftCorner(self):
        return self.__frontTopLeftCorner


    ## @brief Sets the front top left corner.
    #  @param newCorner: (x,y,z) tuple for the front top left corner
    def setFrontTopLeftCorner(self, newCorner):
        self.__frontTopLeftCorner = newCorner


    ## @brief Gets width, height and depth.
    #  @return (w,h,d) tuple containing the width, height and depth.
    def getDimensions(self):
        return self.__dimensions


    ## @brief Sets width, height and depth.
    #  @param dimensions (w,h,d) tuple containing the width, height and depth.
    def setDimensions(self, newSize):
        self.__dimensions = newSize


    ## @brief This function sets the dimensions of the given face.
    #  @param face Face of the box: FRONT or TOP or SIDE.
    #  @return (width, height)
    def getDimensionsOfFace(self, face):
        if face == Box3D.FRONT:
            width,height = self.__dimensions[0],self.__dimensions[1] # w,h
        elif face == Box3D.TOP:
            width,height = self.__dimensions[0],self.__dimensions[2] # w,d
        elif face == Box3D.SIDE:
            width,height = self.__dimensions[1],self.__dimensions[2] # h,d
        else:
            raise ValueError(”Invalid ‘face‘ argument”)
        return width,height


    ## @brief This function calculates the area of the given face.
    #  @param face Face of the box: FRONT or TOP or SIDE.
    #  @return Area of the given face.
    def getAreaOfFace(self, face):
        width,height = self.getDimensionsOfFace(face)
        return width * height
```

```python
    ## @brief This function calculates the perimeter of the given face.
    #  @param face Face of the box: FRONT or TOP or SIDE.
    #  @return Perimeter of the given face.
    def getPerimeterOfFace(self, face):
        width,height = self.getDimensionsOfFace(face)
        return 2 * (width + height)


    ## @brief This function calculates the volume of the rectangle.
    #  @return Volume of the rectangle.
    def getVolume(self):
        width,height,depth = self.__dimensions
        return width * height * depth


    ## @brief This function calculates the surface area of the rectangle.
    #  @return Surface area of the rectangle.
    def getSurfaceArea(self):
        surfaceArea = 0
        for face in (Box3D.FRONT, Box3D.TOP, Box3D.SIDE):
            surfaceArea += (2 * self.getAreaOfFace(face))
        return surfaceArea


## @brief Example of class usage.
if __name__ == '__main__':
    b = Box3D( (1,1,1), (2,2,2) )
    print b.getSurfaceArea()
```

other appendicies
....

B Code for testCircles.c

```
## @file pointADT.py
#  @author Gurankash Singh
#  @brief Provides the PointT ADT class for representing 2D points
#  @date 30 Jan 2017

from math import *

##@brief An ADT that respresents a 2D point
class pointT:

    ## @brief PointT constructor
    #  @details Initializes a PointT object with a cartesian coordinate
    #  @param x The x coordinate of the point
    #  @param y The y coordinate of the point
    def __init__(self, x, y):
        self.__xc = x
        self.__yc = y

    ## @brief Gets the x coordinate of the point
    #  @return The x coordinate of the point
    def xcoord(self):
        return self.__xc

    ## @brief Gets the y coordinate of the point
    #  @return The y coordinate of the point
    def ycoord(self):
        return self.__yc

     ## @brief Determines the distance between 2 points
     #  @details Uses pythagorean theorem
     #  @param p Another point
     #  @return The distance between the given points
    def dist(self, p):
        xDist = self.__xc - p.xcoord()
        yDist = self.__yc - p.ycoord()
        return sqrt(xDist ** 2 + yDist ** 2)
```

C Makefile

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

SRC = Box3D.py

.PHONY: all prog doc clean

prog:
        $(PY) $(PYFLAGS) $(SRC)

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

all: prog doc

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

D Partner's Code

```
## @file pointADT.py
#  @author Gurankash Singh
#  @brief Provides the PointMass ADT class for representing point masses
#  @date 30 Jan 2017


from PointADT import *

##@brief An ADT that respresents a 2D point
class pointMassT:

    UNIVERSAL_G = 6.672e-11

    ## @brief PointMassT constructor
    #  @details Initializes a PointMassT object with a point and corresponding mass value
    #  @param p The coordinates of the point type pointT
    #  @param m The mass value
    def __init__(self, p, m):
        self.__pt = p
        self.__ms = m

    ## @brief Gets the coordinates of the point
    #  @return The point of type pointT
    def point(self):
        return self.__pt

    ## @brief Gets the mass the point
    #  @return The the mass value
    def mval(self):
        return self.__ms

     ## @brief Determines the force between 2 point masses
     #  @details Uses the force formula with UniversalG constant
     #  @param p Another point mass
     #  @return The force between the given point masses
    def force(self, p):
        m = self.__ms * p.mval()
        r = dist(self.__pt, point(p))
        return UNIVERSAL_G * m / (r ** 2)

     ## @brief Determines the force vector between the 2 point masses
     #  @details Uses the force vector formula
     #  @param p Another point mass
     #  @return The force vector
    def Fx(self, p):
        xDist = point(p).xcoord() - self.__pt.xcoord()
        r = dist(self.__pt, point(p))
        return self.__force(p) * (xDist/r)
```

This code can also be found at the following link:

Link