'''

Keni Mou

Kmou1@binghamton.edu

Assignment10 Exercise 1

Lab Section 52

CA Kyle Mille

'''

#Constants

NUM1=54

NUM2=42

DEN1=12

DEN2=24

class Fraction:

# --------------------------------------------------------------------------

# Constructor

# param top (int)

# param bottom (int)

def \_\_init\_\_(self,top,bottom):

self.\_\_num = top #the numerator is on top

self.\_\_den = bottom #the denominator is on the bottom

# --------------------------------------------------------------------------

# Accessors

#return numerator

def getNum(self):

return self.\_\_num

#return denominator

def getDen(self):

return self.\_\_den

#get the common factor

def \_\_gCD(self):

m = self.\_\_num

n = self.\_\_den

while m % n != 0:

oldm = m

oldn = n

m = oldn

n = oldm% oldn

return n

#to invoke +

#param otherfraction(Fraction)

#return the sum

def \_\_add\_\_(self,otherfraction):

return self.add(otherfraction)

#param otherfraction(Fraction)

#return sum of two fractions

def add(self, otherfraction):

newnum = self.\_\_num \* otherfraction.getDen() + self.\_\_den \* otherfraction.getNum()

newden = self.\_\_den \* otherfraction.getDen()

fraction = Fraction(newnum, newden)

fraction.simplify()

return fraction

# param otherfraction(Fraction)

# return ture if both fraction are equal to each other, false otherwise.

def isEqualTo(self, otherfraction):

self.simplify()

otherfraction.simplify()

return self.\_\_den() == otherfraction.getDen and self.\_\_num == otherfraction.getNum()

# --------------------------------------------------------------------------

# Mutators

#simplified fraction

def simplify(self):

common = self.\_\_gCD()

self.\_\_num = self.\_\_num // common

self.\_\_den = self.\_\_den // common

# --------------------------------------------------------------------------

# toString

def \_\_str\_\_(self):

return str(self.\_\_num) + "/" + str(self.\_\_den)

# ----------------------------------------------------------------------------

def main():

fraction1=Fraction(NUM1,DEN1)

fraction2=Fraction(NUM2,DEN2)

print("Fraction1:%s , Fraction2:%s"%(fraction1, fraction2))

print("Test getNum for fraction1", fraction1.getNum())

print("Test getNum for fraction2", fraction2.getNum())

print("Test getDen for fraction1", fraction1.getDen())

print("Test getDen for fraction2", fraction2.getDen())

print("Test add", fraction1.add(fraction2))

print("Test \_\_add\_\_", fraction1+fraction2)

fraction1.simplify()

fraction2.simplify()

print("Test simplify", fraction1,fraction2)

main()

Python 3.3.3 (v3.3.3:c3896275c0f6, Nov 18 2013, 21:19:30) [MSC v.1600 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

Fraction1:54/12 , Fraction2:42/24

Test getNum for fraction1 54

Test getNum for fraction2 42

Test getDen for fraction1 12

Test getDen for fraction2 24

Test add 25/4

Test \_\_add\_\_ 25/4

Test simplify 9/2 7/4

>>>

'''

Keni Mou

Kmou1@binghamton.edu

Assignment10 Exercise 2

Lab Section 52

CA Kyle Mille

'''

from point import Point

#CONSTANTS

DOUBLE=2

HALF=0.5

LOC\_POINT\_X=3

LOC\_POINT\_Y=5

WIDTH=10

HEIGHT=20

CHE\_POINT\_X=4

CHE\_POINT\_Y=25

class Rectangle:

"""Rectangle class using Point, width and height"""

# --------------------------------------------------

# Constructor

# param initP(Point)

# param initW(floar)

# param initH(floar)

def \_\_init\_\_(self,initP,initW,initH):

self.\_\_location = initP

self.\_\_width = initW

self.\_\_height = initH

# ----------------------------------------------------

# Accessors

# return width

def getWidth(self):

return self.\_\_width

# return height

def getHeight(self):

return self.\_\_height

# return perimeter

def perimeter(self):

return self.\_\_width\*DOUBLE + self.\_\_height\*DOUBLE

# return diagonal

def diagonal(self):

diagonal = (self.\_\_width\*\*DOUBLE + self.\_\_height\*\*DOUBLE)\*\*HALF

return diagonal

# return area

def area(self):

return self.width \* self.height

# param point(Point)

# return true if the point is inside the rectangle

def encloses(self, point):

return point.getX()>self.\_\_location.getX() and \

point.getX() < (self.\_\_location.getX()+ self.\_\_width) and \

point.getY()>self.\_\_location.getY() and \

point.getY() < (self.\_\_location.getY()+ self.\_\_height)

# ---------------------------------------------------

# Mutators

# transpose

def transpose(self):

temp = self.\_\_width

self.\_\_width = self.\_\_height

self.\_\_height = temp

# ---------------------------------------------

# toString

def \_\_str\_\_(self):

return "Point: %s, Width: %s, Height: %s"% \

(self.\_\_location, self.\_\_width, self.\_\_height)

#----------------------------------------------------------

def main():

#create a rectangle object

rectangle=Rectangle(Point(LOC\_POINT\_X,LOC\_POINT\_Y),WIDTH,HEIGHT)

point=Point(CHE\_POINT\_X,CHE\_POINT\_Y)

print("rectangle:",rectangle)

print("Test getWidth and getHeight")

print("Width: %s, Height: %s"% (rectangle.getWidth(),rectangle.getHeight()),'\n')

print("Test perimeter")

print(rectangle.perimeter(),'\n')

print("Test transpose")

rectangle.transpose()

print(rectangle,'\n')

print("Test diagonal")

print(rectangle.diagonal(),'\n')

print("Test encloses, point:%s"%(point))

print(rectangle.encloses(point))

main()

'''

Keni Mou

Kmou1@binghamton.edu

Lab10 Exercise 1

Lab Section 52

CA Kyle Mille

'''

import math

# Point class for representing and manipulating

# x,y coordinates.

class Point:

# --------------------------------------------------------------------------

# Constructor

# param initX (int)

# param initY (int)

def \_\_init\_\_(self, initX = 0, initY = 0):

self.\_\_x = initX

self.\_\_y = initY

# --------------------------------------------------------------------------

# Accessors

# return value of x (int)

def getX(self):

return self.\_\_x

# return value of y (int)

def getY(self):

return self.\_\_y

# return distance from origin (float)

def computeDistanceFromOrigin(self):

return ((self.\_\_x \*\* 2) + (self.\_\_y \*\* 2)) \*\* 0.5

# return distance from other point (float)

def computeDistanceFromPoint(self, otherP):

dx = (otherP.getX() - self.\_\_x)

dy = (otherP.getY() - self.\_\_y)

return math.sqrt(dy\*\*2 + dx\*\*2)

# return x reflection (Point)

def createXReflection(self):

return Point(self.\_\_x, self.\_\_y \* -1)

## param otherPoinst(Point)

# return slope to point (float)

# Note exception handling!

def computeSlopeFromPoint(self, otherPoint):

try:

slope = (self.\_\_y - otherPoint.getY()) / (self.\_\_x - otherPoint.getX())

except ZeroDivisionError:

slope = None

return slope

# return slope from origin (float)

def computeSlopeFromOrigin(self):

slope = self.computeSlopeFromPoint(Point(0, 0))

return slope

## param ohterPoint(Point)

# return y-intercept of line (float)

def computeYIntercept(self, otherPoint):

return self.\_\_y - (self.computeSlopeFromPoint(otherPoint) \* self.\_\_x)

## param ohterPoint(Point)

# return coefficients of line (tuple)

def getLineTo(self, otherPoint):

slope = self.computeSlopeFromPoint(otherPoint)

intercept = self.computeYIntercept(otherPoint)

return (slope, intercept)

# --------------------------------------------------------------------------

# Mutators

# param dx (int)

# param dy (int)

def move(self, dx, dy):

self.\_\_x += dx

self.\_\_y += dy

# --------------------------------------------------------------------------

# toString

# return string representation of Point object (str)

def \_\_str\_\_(self):

return "(x:%d, y:%d)" % (self.\_\_x, self.\_\_y)

Python 3.3.3 (v3.3.3:c3896275c0f6, Nov 18 2013, 21:19:30) [MSC v.1600 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

rectangle: Point: (x:3, y:5), Width: 10, Height: 20

Test getWidth and getHeight

Width: 10, Height: 20

Test perimeter

60

Test transpose

Point: (x:3, y:5), Width: 20, Height: 10

Test diagonal

22.360679774997898

Test encloses, point:(x:4, y:25)

False

>>>