import time, math, os, pickle

import tkFont

import turtle

class trigo:

@staticmethod

def trigo\_main(q):

#gives colour in rgb mode with scale 255

turtle.colormode(255)

turtle.pencolor((197, 83, 253))

turtle.width(1.4)

turtle.penup()

turtle.goto(-250,0)

turtle.pendown()

# i have multiplied by 10 in x and y coordinate so that graph looks bigger and readable and shifted actual axis by 250 pixles to make whole graph visible

for i in range (50):

if q=="cos":

turtle.goto((i\*10-250),10\*math.cos(i))

if q=="sin":

turtle.goto((i\*10-250),10\*math.sin(i))

turtle.penup()

b = raw\_input("Enter where you want to shift graph\n\t1.Nowhere\n\t2.Right\n\t3.Left\n")

trigo.trigo\_shift(b,q)

@staticmethod

def trigo\_axis\_make():

turtle.goto(0,0)

turtle.pendown()

turtle.pencolor("black")

#making X coordinates

turtle.goto(-300,0)

turtle.goto(300,0)

turtle.goto(0,0)

#making Y coordinates

turtle.goto(0,20)

turtle.goto(0,-20)

turtle.goto(0,0)

turtle.penup()

@staticmethod

def trigo\_mark\_axis():

turtle.pencolor("blue")

turtle.width(1)

#marking Y coordinates

#marks 1 on yaxis

turtle.penup()

turtle.goto(-5,10)

turtle.pendown()

turtle.write("1",font=("Arial",6,"normal"))

#marks -1 on yaxis

turtle.penup()

turtle.goto(-7,-20)

turtle.pendown()

turtle.write("-1",font=("Arial",6,"normal"))

#marking x coordinate

turtle.colormode(255)

#let us obtain color in rgb color codes which has code range of each color of 255

turtle.pencolor((103, 150, 8))

turtle.width(1)

for i in range (16):

if i==8:

# to write for 0 and other as it will print 0pi then it is i=8 bcoz i have shifted graph toward left by 250 pixles to make whole graph come in window and 8\*pi\*10 will give value around 251

turtle.penup()

turtle.goto(2,-10)

turtle.pendown()

turtle.write("0",font=("Arial",6,"bold"))

else:

#marks a kink in graph

turtle.penup()

turtle.goto((math.pi\*i\*10)-(250),+4)

turtle.pendown()

turtle.goto((math.pi\*i\*10)-(250),-4)

turtle.penup()

#writes the value

turtle.goto((math.pi\*i\*10)-(250),-20)

turtle.pendown()

turtle.write((str(i-8)+("pi")) ,font=("Arial",6,"bold"))

@staticmethod

def trigo\_shift(b,q):

turtle.colormode(255)

turtle.pencolor((240, 169, 108))

turtle.width(1)

c = input("Enter the value till you want to shift if you want to shift\nright\nleft\nnowhere")

d = c\*10

if b.lower() in ["right"]:

#to get graph lying on same axis

for i in range (-60,60):

if -251<(i\*10-250+d)<251:

if q=="cos":

turtle.goto((i\*10-250+d),10\*math.cos(i))

if q=="sin":

turtle.goto((i\*10-250+d),10\*math.sin(i))

turtle.pendown()

elif b.lower() in ["left"]:

for i in range (-60,60):

if -251<(i\*10-250-d)<251:

if q=="cos":

turtle.goto((i\*10-250-d),10\*math.cos(i))

if q=="sin":

turtle.goto((i\*10-250-d),10\*math.sin(i))

turtle.pendown()

elif b.lower() in ["nowhere", "no"]:

turtle.penup()

turtle.goto(0,0)

else:

turtle.exitonclick()

exit()

class circle:

turtle.colormode(255)

turtle.pencolor((240, 169, 108))

turtle.width(3.2)

@staticmethod

def drawCircle (centerpoint, radius):

(x,y) = centerpoint

turtle.up()

turtle.setpos(x,y)

turtle.down()

#r\*10 so that circle looks bigger

turtle.circle(radius\*10)

class cycloid:

@staticmethod

def cycloid():

turtle.penup()

turtle.goto(-300,0)

turtle.pendown()

turtle.colormode(255)

turtle.pencolor((240, 169, 108))

turtle.width(3.2)

turtle.speed(1000)

for i in range (400):

#this is equation of cycloid

turtle.goto(15\*(i/10-math.sin(i/10))-300,15\*(1-math.cos(i/10)))

class regular\_polygon:

@staticmethod

def polygon(a):

if a>2:

turtle.pendown()

for i in range(a+1):

turtle.forward(50)

# the angle of each vertice of a regular polygon is 360 divided by the number of sides

# turtle.left rotates turtle at its position by dgrees given in argument

turtle.left(360/a)

else:

print "invalid input"

exit()

class hypocycloid:

@staticmethod

def hypo(k):

r=50

#k is no. of cusps.........Cusp- A sharp point on a curve.........If k is a rational number, say k = p/q expressed in simplest terms, then the curve has p cusps.

#If k is an irrational number, then the curve never closes, and fills the space between the larger circle and a circle of radius R ? 2r

turtle.colormode(255)

turtle.pencolor((74, 185, 240))

turtle.width(7)

#have taken radius divided by 10 as in circle class have multiplied incoming radius into 10 to make circle bigger

circle.drawCircle((0,-r\*k),r\*k/10)

turtle.penup()

turtle.goto(r\*k,0)

turtle.pendown()

turtle.speed(120)

turtle.colormode(255)

turtle.pencolor((75, 201, 142))

turtle.width(.8)

for i in range(0,1000):

turtle.goto(r\*(k-1)\*math.cos(i)+r\*math.cos((k-1)\*i),r\*(k-1)\*math.sin(i)-r\*math.sin((k-1)\*i))

turtle.penup()

turtle.goto(-300,-r\*k-30)

turtle.pendown()

turtle.pencolor("black")

turtle.width(2)

turtle.write("Quick Fact: \nfor k=2 the curve is a straight line and the circles are called Cardano circles.\nThey are used in high-speed printing",font=("Arial",8,"normal"))

class epicycloid:

@staticmethod

def epicycloid(k):

r=50

turtle.colormode(255)

turtle.pencolor((74, 185, 240))

turtle.width(7)

circle.drawCircle((0,-r\*k),r\*k/10)

turtle.penup()

#as at i=0--in equation of epicycloid,turtle is at (r\*k,0)

turtle.goto(r\*k,0)

turtle.pendown()

turtle.speed(120)

turtle.colormode(255)

turtle.pencolor((75, 201, 142))

turtle.width(.8)

for i in range(0,1000):

turtle.goto(r\*(k+1)\*math.cos(i)-r\*math.cos((k+1)\*i),r\*(k+1)\*math.sin(i)-r\*math.sin((k+1)\*i))

class polynomial:

@staticmethod

def new\_polynomial():

s=[]

a = input("Degree of polynomial\n")

for i in range (a+1):

b = input("Enter coefficient of power "+ str(i)+ "\t")

s+=[b]

return s

@staticmethod

def poly\_make\_axis():

turtle.colormode(255)

#let us obtain color in rgb color codes which has code range of each color of 255

turtle.pencolor((103, 150, 8))

turtle.width(1)

#x axis

turtle.goto(-300,0)

turtle.goto(300,0)

turtle.goto(0,0)

#y axis

turtle.pendown()

turtle.goto(0,250)

turtle.goto(0,-250)

turtle.penup()

turtle.goto(0,0)

@staticmethod

def poly\_main(s):

turtle.colormode(255)

turtle.pencolor((240, 169, 108))

turtle.width(1.75)

for j in range(-40,40):

p=0

#s(list)is of form {2,1,2,-3}.......meaning coefficient of x^0 is 2,x^1 is 1,x^2 is 2,x^3 is -3

for i in range(len(s)):

p+=s[i]\*(j\*\*i)

#here p becomes the polynomial... -3(i^3)+2(i^2)+1(i^1)+2(i^0)

#in which i is varied to make graph

#to contain graph in window

if -290<p<290:

turtle.goto(j\*10,p)

turtle.pendown()

if p==0:

#if by chance, a root comes integer in -40 to 40...it is marked

turtle.penup()

turtle.goto(j\*10,0)

turtle.pendown()

#make kink

turtle.goto(j\*10,-4)

turtle.goto(j\*10,+4)

turtle.penup()

#write the integer root....try drawing (x-2)^3={-8,12,-6,1}..(values to be put according to order when program is run

turtle.goto(j\*10,-15)

turtle.pendown()

turtle.write(j)

turtle.penup()

turtle.goto(j\*10,0)

turtle.pendown()

#and graph is continued from this pt again

def new():

y=raw\_input("What do you want to draw -\n1.Cos\n2.Sin\n3.Circle\n4.Regular polygon\n5.Polynomial\n6.Cycloid\n7.Hypocycloid\n8.Epicycloid\n")

if y.lower() in ["1", "cos"]:

trigo.trigo\_axis\_make()

trigo.trigo\_mark\_axis()

trigo.trigo\_main("cos")

var = 0

if y.lower() in ["2", "sin"]:

trigo.trigo\_axis\_make()

trigo.trigo\_mark\_axis()

trigo.trigo\_main("sin")

var = 0

if y.lower() in ["3", "circle"]:

a = input("Enter x coordinate of circle\t")

b = input("Enter y coordinate of circle\t")

r = input("Enter radius\t")

circle.drawCircle((a,b),r)

var = (a,b,r)

if y.lower() in ["4", "regular polygon"]:

a = input("Enter number of sides of polygon\t")

regular\_polygon.polygon(a)

var = a

if y.lower() in ["5", "polynomial"]:

a = polynomial.new\_polynomial()

polynomial.poly\_make\_axis()

polynomial.poly\_main(a)

var = a

if y.lower() in ["6", "cycloid"]:

cycloid.cycloid()

var = 0

if y.lower() in ["7", "hypocycloid"]:

k = input("Enter no. of cusps\t")

#to make cardano line--k=2

#to make deltoid---k=3

#to make asteroid--k=4

#to make star---k=5

hypocycloid.hypo(k)

var = k

if y.lower() in ["8", "epicycloid"]:

k = input("Enter no. of cusps\t")

#for best result---k=2 or 3 or 5

epicycloid.epicycloid(k)

var = k

return (y,var)

def prev(y, var):

if y.lower() in ["1", "cos"]:

trigo.trigo\_axis\_make()

trigo.trigo\_mark\_axis()

trigo.trigo\_main("cos")

if y.lower() in ["2", "sin"]:

trigo.trigo\_axis\_make()

trigo.trigo\_mark\_axis()

trigo.trigo\_main("sin")

if y.lower() in ["3", "circle"]:

(a,b,r) = var

circle.drawCircle((a,b),r)

if y.lower() in ["4", "regular polygon"]:

a = var

regular\_polygon.polygon(a)

if y.lower() in ["5", "polynomial"]:

a = var

polynomial.poly\_make\_axis()

polynomial.poly\_main(a)

if y.lower() in ["6", "cycloid"]:

cycloid.cycloid()

if y.lower() in ["7", "hypocycloid"]:

k = var

hypocycloid.hypo(k)

if y.lower() in ["8", "epicycloid"]:

k = var

epicycloid.epicycloid(k)

while True:

# Asking to continue with the program

v = raw\_input("Do you want to plot new graph or from last function plotted -\n1.New Graph\n2.Previous Graph\n3.Exit\n")

turtle.clear()

turtle.penup()

turtle.goto(0,0)

turtle.pendown()

# Allowing multiple possible inputs to be accepted

if v.lower() in ["new graph", "new", "1"]:

# y,var are the variables with the information regarding the graph needed to reproduce it

(y,var) = new()

# Checking if Graphs folder already there

if not os.path.exists(".\\Graphs"):

# Making said folder if not found

os.mkdir(".\\Graphs")

# Using a generator function to get a list of files in Graphs folder

(root,dir,files) = os.walk(".\\Graphs").next()

# Calculating the name of the new file

filenum = len(files)

# Opening a new file with .graph extension which is self created but can still be opened in binary mode

graph = file(".\\Graphs\\"+str(filenum+1)+".graph", "wb")

# Filling the file with required data

pickle.dump((y,var), graph)

# Remember to close to make sure everything functions smoothly!

graph.close()

# Again allowing multitudes of possible inputs

elif v.lower() in ["2", "previous", "previous graph"]:

# Getting the filelist

(root, dir, files) = os.walk(".\\Graphs").next()

# Sorting so as to be able to determine the last file

files.sort()

# Opening the file

previous = file(".\\Graphs\\" + files[-1], "rb")

# Getting the critical variables required to make graph

(y,var) = pickle.load(previous)

# Closing to ensure smooth functioning

previous.close()

# Deleting opened graph

os.remove(".\\Graphs\\"+files[-1])

# Sending acquired graph information for processing

prev(y, var)

elif v.lower() in ["3", "exit"]:

turtle.exitonclick()

exit()

else:

print "Make an appropriate choice"

print "The program now closes"

time.sleep(1)

exit()

turtle.exitonclick()