DREXEL UNIVERSITY
Electrical and Computer Engineering
College of Engineering

ECEC 301
Programming for Engineers

Programming Assignment 2: Spirograph

Professor: James A. Shackleford

Section 062

Yonatan Carver
yac25@drexel.edu

Due Date: October 24, 2017

Introduction

The goal of this programming assignment is to create a virtual spirograph using Python 2.7. A spirograph is a toy that is used to draw geometric mathematical curves. The Python module "`turtle`" is an add-on for the Python module "`tkinter`" which allows the user to create and graphically display intricate shapes and patterns.

Theory

Physical spirographs typically have two geared rings, one outer ring and one inner ring. The user puts their writing utensil through a hole in the inner ring (dark circle in *figure 1*) and as they move the inner gear around the outer gear, a geometric pattern is produced on a piece of paper.

Spirographs produce many types of curves including hypotrochoids and epitrochoids. A hypotrochoid is a roulette (a kind of curve) created by tracing a point that is some distance d from a smaller circle (radius r) traveling around the interior of a larger, fixed circle of radius R (*figure 2*). Additionally, an epitrochoid is a roulette created by tracing a point on a small circle with radius r travelling around the exterior of a larger circle with radius R (*figure 3*). The spirographs drawn in this project resemble these two types of roulettes.

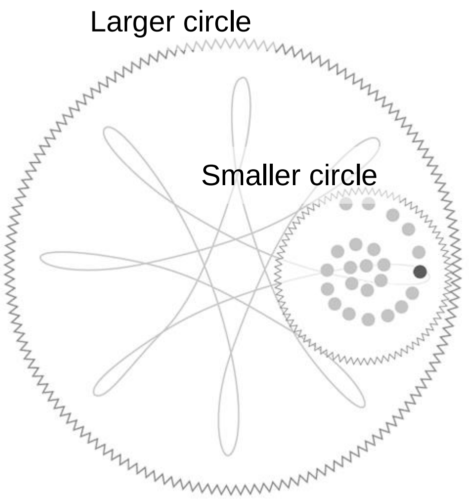

Figure 1: Spirograph Components

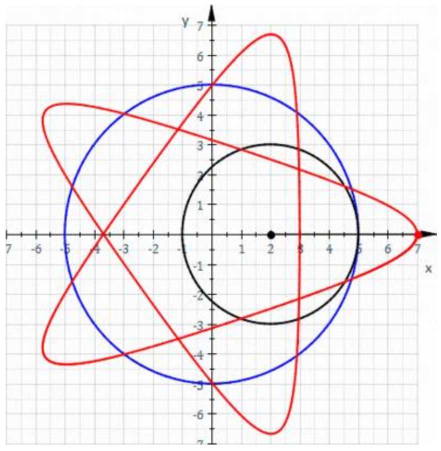

Figure 2: Hypotrochoid Roulette

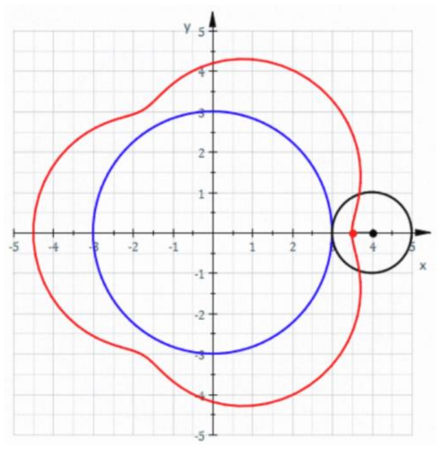

Figure 3: Epitrochoid Roulette

The movement of a spirograph can be represented mathematically by the two-dimensional vector P (*equation* 1). The x and y coordinates of a spirograph, at any angle (θ) are represented in *equation 2* and *3*.

$$\vec{P}(\theta) = \left[P_x(\theta), P_y(\theta)\right] \quad (1)$$

$$P_x(\theta) = R\left((1-k)\cos(\theta) + l\,k\cos\left(\frac{1-k}{k}\theta\right)\right) \quad (2)$$

$$P_y(\theta) = R\left((1-k)\cos(\theta) - l\,k\cos\left(\frac{1-k}{k}\theta\right)\right) \quad (3)$$

Table 1: Symbol Descriptions

Symbol	Meaning
R	Radius of larger circle
r	Radius of smaller circle
k	Ratio of larger circle to smaller circle (r: R)
θ	Angle at any angle (θ)
l	Ratio of pen distance to center of smaller circle

A question that might arise while drawing a spirograph is: how does the drawer know when the full spirograph is complete, i.e. when the marker returns to its original starting position? In almost every instance, the writing utensil will make multiple complete cycles around the larger circle before the drawing is complete. One must calculate the period of the spirograph (the period is the number of rotations the spirograph must take before returning to its home position). The ratio can be calculated by observing the ratio between the smaller circle radius, r, and the larger circle radius, R (*equation 4*). To calculate the period, the numerator (smaller circle radius, r) is then divided by the greatest common denominator of both radii (*equation 5*).

$$radius\,ratio = \frac{r}{R} \quad (4)$$

$$period = T = \frac{r}{\gcd(r, R)} \quad (5)$$

To create graphical representations of these spirographs, the Python module "`turtle`" is used. `Turtle` is a graphics add-on for "`tkinter`," a graphical user interface package for Python. In the most basic sense, `turtle` allows the user to create a "`Turtle`" object that can receive coordinates and draw a line behind it along its path. In this case, the turtle moves to the coordinates specified in *equation 2* and *3* and a spirograph is generated.

Additionally, `turtle` offers methods that provide the user with options to change the turtle line color, linewidth, speed, starting position, direction, and many more. (For official documentation on the `turtle` module, see appendix 1)

Implementation

Before creating the class that is Spirograph, the following modules must be imported to maximize speed and efficiency while running the script: `turtle`, `math` (`sin`, `cos`, `radians`), and `fractions` (`gcd`).

`__init__(self, R)`

The magic method `__init__` function runs as soon as the `Spirograph` class is called. This function takes one argument, R, which is the large circle radius. Additionally, the window in which the spirograph is displayed (lines 4 - 7) and turtle objects that draws the spirograph (lines 8 - 11) are created here. Line 12 sets a variable called `self._R` equal to the user inputted value for the larger circle radius. It is important to note that all variables preceded by "`self`" in a function can be accessed throughout the entire class so long as "`self`" is passed as the first argument in each function.

```
1       def __init__(self, R):
2           """ create a new spirograph toy with outer radius R """
3
4           self._window = turtle.Screen()      # create screen
5           self._window.screensize(canvwidth=500, canvheight=500, bg='black')     #
6   set window dimensions & bgcolor
7           self._window.title("Yoni's Turtle") # title of window
8           self._lloyd = turtle.Turtle()       # initiate Turtle (turtle's name is
9   Lloyd!)
10          self._lloyd.speed(0)                # set turtle speed
11          self._lloyd.shape('turtle')         # display turtle icon
12          self._R = int(R)                    # input: Radius (R)
```

`setSmallCircle(self, r)`

This function simply receives a user input for the small circle radius.

```
     def setSmallCircle(self,r):
         """ set the radius of the small circle used to draw """

         self._r = int(r)                    # input: radius (r)
```

`setPen(self, r, color)`

This function takes in two arguments: l and color. "l" is the ratio of the pen distance to the center of the smaller circle and "color" is the pen color. By default, the turtle is set to automatically draw a line whenever moved. To ensure no extraneous lines are drawn, the method, pen up, "`.pu()`" is called on our instance of turtle (line 6) – this method lifts up the pen and will not draw a line until pen down, "`.pd()`", is called. We are able to call "`.pu()`", "`.pencolor()`", and "`.pensize()`" on "`self._lloyd`" because we created this variable and instantiated an instance of turtle in the `__init__` function (line 8).

```
1       def setPen(self, l, color):
2           """ set the pen color and its distance from C """
3
4           self._l = l                        # input: l (ratio of pen distance to
    center of smaller circle)
5
6           self._lloyd.pu()                    # pick pen up
7
8           self._lloyd.pencolor(color)         # input: pen color
9           self._lloyd.pensize(2)              # set pen size
```

`draw(`*`self`*`)`

This function is where the spirograph is drawn. Additionally, this function holds all mathematical equations outlined in the Theory section of this report. Lines 4 – 6 calculate the greatest common denominator and ratio of r and R – all items needed to compute the period of the spirograph (*equation 5*). Line 7 is a quick way to assign three values to three variables – R (outer circle radius), k (ratio of r to R), and l (ratio of the pen distance to the center of the smaller circle).

The spirograph is calculated and drawn in the for loop on lines 8 – 14. The range of the for loop is 0 to 360° multiplied by the number of complete cycles needed to create the full spirograph. Line 10 converts the angle to radians that can be used with *equations 2* and *3*, represented on lines 11 and 12. Line 13 moves the turtle to the vector P and line 14 puts the pen down to draw a line. To clean up the final image of the spirograph, the turtle object is hidden at the end of the draw method.

```
1       def draw(self):
2           """ draw a spirograph using the current small circle and pen settings """
3
4           _gcd = gcd(self._r, self._R)                                  # find gcd of r and R (gcd
5   not needed anywhere else in class so it is a local variable)
6           self._k = float(self._r)/float(self._R)                     # ratio of r to R as a float
7           R, k, l = self._R, self._k, self._l                         # assign variables
8           for i in range(0, 360*(self._r/_gcd)):                        # for every angle in range
9   period to complete one full spirograph...
10              theta = radians(i)                                      # theta in radians
11              self._Px = R*((1-k)*cos(theta) + l*k*cos((1-k)*theta/k))    # x location pen
12              self._Py = R*((1-k)*sin(theta) - l*k*sin((1-k)*theta/k))    # y location of pen
13              self._lloyd.setpos(self._Px, self._Py)                # move Lloyd to x and y location
14              self._lloyd.pd()                                      # put pen down & draw spirograph
17          self._lloyd.hideturtle()                                # hide Lloyd at the end of drawing
```

`clear(`*`self`*`)`

This function wipes the screen clean using a built-in function of `turtle.Screen()`, which was assigned the name "`self._window`" in line 4 of the `__init__` function. This function essentially creates a blank slate that is ready to accept another Spirograph drawing instance.

```
1       def clear(self):
2           """ reset the drawing surface """
3
4           self._window.clearscreen()          # clear screen
```

Testing the Spirograph

Below is a script that can be used to test the spirograph class. Line 1 imports the Spirograph class from the file outlined above. Line 5 assigns variable `my_spirograph` to the function with a large radius of 500. The argument ("500," in this case) entered when initiating the Spirograph on line 5 is passed through the `__init__` function which gets assigned to the variable "`self._R.`" Lines 8, 13, and 21 call the "`setSmallCircle`" function which assigns the input to the small circle radius. Lines 9, 14, and 22 call the "`setPen`" function that sets the distance of the pen from the center of the smaller circle and the color of the pen. Lines 10, 15, and 23 run the "`.draw`" function of "`turtle`" which actually draws the spirograph. Line 18 clears the drawing, window, and any patterns that were previously created.

```
1   from spirograph import Spirograph as spirograph
2
3   # Create a new Spirograph toy with R = 500
4
5   my_spirograph = spirograph(500)
6
```

```
# Draw one curve
my_spirograph.setSmallCircle(85)
my_spirograph.setPen(0.65, 'red')
my_spirograph.draw()

# ...and then draw another on top of the first
my_spirograph.setSmallCircle(120)
my_spirograph.setPen(0.22, 'blue')
my_spirograph.draw()

# ...and then get a new sheet of paper
my_spirograph.clear()

# ...and draw another
my_spirograph.setSmallCircle(20)
my_spirograph.setPen(0.8, 'purple')
my_spirograph.draw()
```

Testing Results + Screenshots

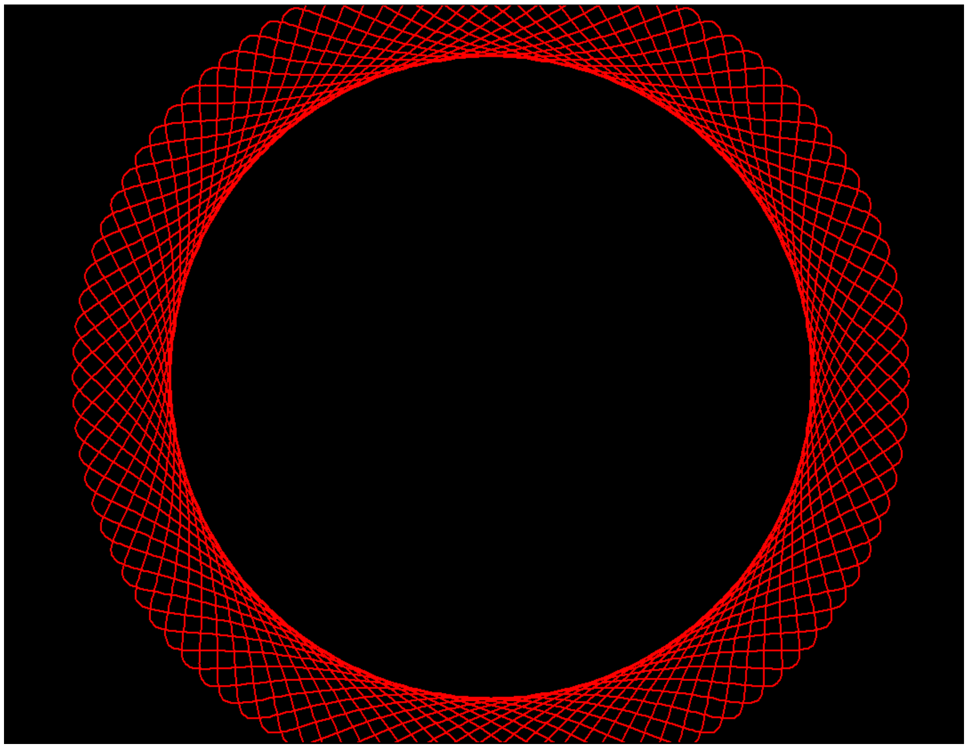
*Figure 4: Spirograph - Test 1**

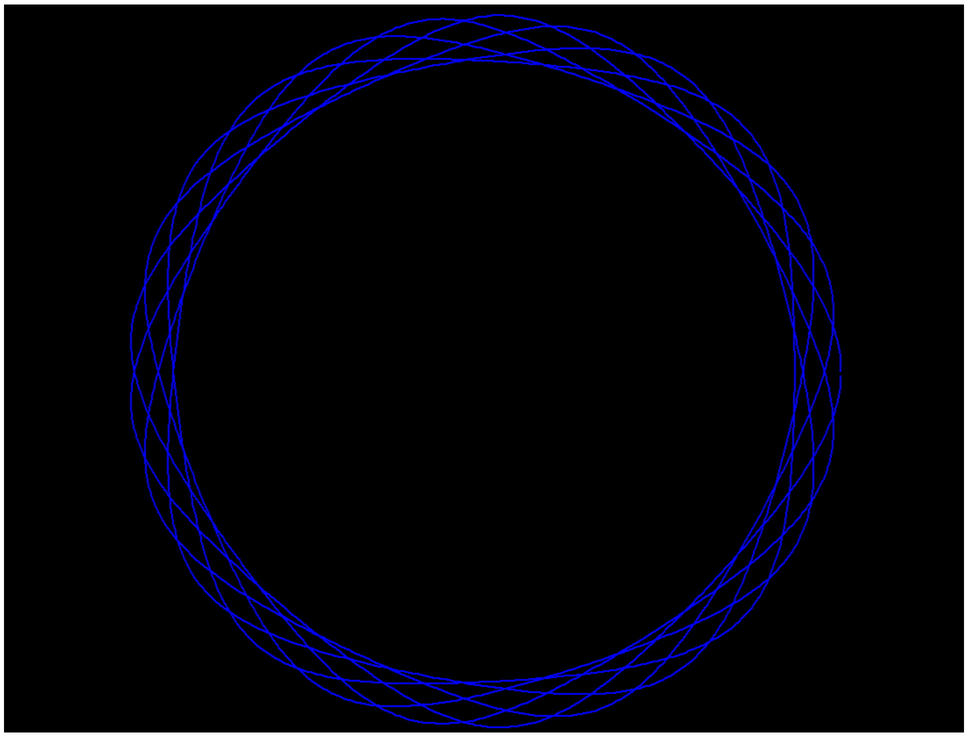

Figure 5: Spirograph - Test 2

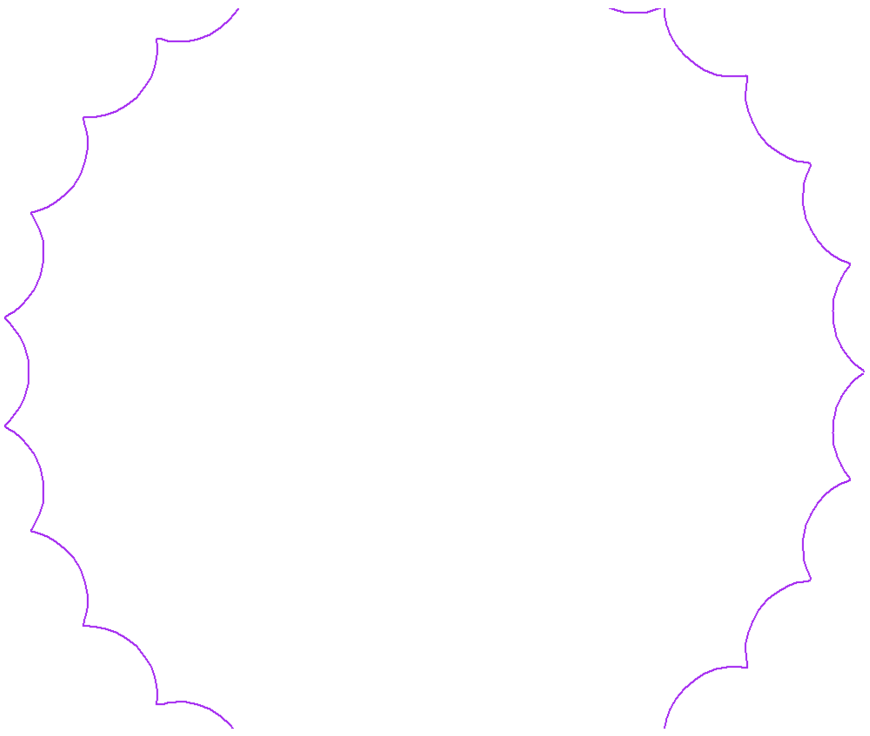

*Figure 6: Spirograph - Test 3**

* Unfortunately, due to the size of the window the test script was run in, the top and bottom portions of the spirograph were cut off.

Appendix

1) https://docs.python.org/2/library/turtle.html