Homework 3 – Fractal Designs – Due Monday, Nov. 6

In Lab 10, you will write a Python program to draw a recursively generated self-similar shape called a Sierpinski Triangle. In mathematics there are many such shapes that are known as "fractals," and the Wikipedia page on Fractals has a good discussion with different examples (you aren't required to read this, but you might find it interesting). While fractals are often studied simply for the beauty of the drawings, there are practical uses too: at the bottom of the Wikipedia page is an example that shows a fractal construction of an artificial mountain - many game designers use fractals for generating realistic artificial landscapes for games, and due to the self-similarity of the structure the game can zoom in on this artificial structure at whatever scale is needed. In this homework, you will learn about some self-similar shapes that go beyond the Sierpinski Triangle, see some examples of where they occur both in nature and in design, and then experiment in Python with drawing some recursively-generated shapes.

Self-similar shapes and drawings can have a high level of symmetry and infinitely-detailed structure. Many societies and artists have found great beauty in fractal designs, and in particular many African geometric designs that are found in fabrics and in village design are based on fractals. This homework has a couple of initial activities that you should do before getting to the main part of the assignment. I have put dates on these activities to indicate when I will assume you have done them, so we can discuss these items briefly in class. However, you are encouraged to work faster than this so you have plenty of time to think about and work on the Python code.

Activity 1 (finish no later than Friday, October 20): Watch the video of the TED talk "Ron Eglash: The fractals at the heart of African designs" at http://www.ted.com/talks/ron_eglash_on_african_fractals.html . The video is 17 minutes, and you should spend at least 30 minutes thinking about the ideas and exploring fractals online.

Activity 2 (finish no later than Friday, October 27): Go through the online lesson on fractals and design at http://span.uncg.edu/fractals which should take about one hour to complete (that depends on how much time you spend playing with the fractal explorer).

Activity 3 (finish no later than Monday, November 6): You should create Python programs to draw at least two of the fractals from the preceding online lesson, where the two should be chosen from the list at the end of this activity. There is an area set aside in the online Python lab manual (on `scode`) for you to work and save your programs. How much time should this take? It could take anywhere from as little as one hour to as much as twenty very frustrating hours. If you get really stuck, seek help!

What's great about fractal drawings is that it takes very little code to draw very intricate shapes. The bad part is that for such a small amount of code, it can be very difficult to wrap your mind around. Let's walk through how to make a Python program to draw a Koch curve (the first example in the lesson above – the code is available on `scode` for you to run and experiment with). This, addition to the practice you will get

in Lab 10, will give you a couple of examples to look at, think about, and play around with before you have to make your own fractal drawing code.

The idea is fairly straightforward: we pass three parameters, giving the turtle to use for drawing, the size of the drawing, and the number of levels of recursion to use. The base case is a 0 level Koch curve, and will just draw a single straight line of the specified length for the base case. At higher levels we make four segments, each of which is a Koch curve at a smaller level. Here's the code:

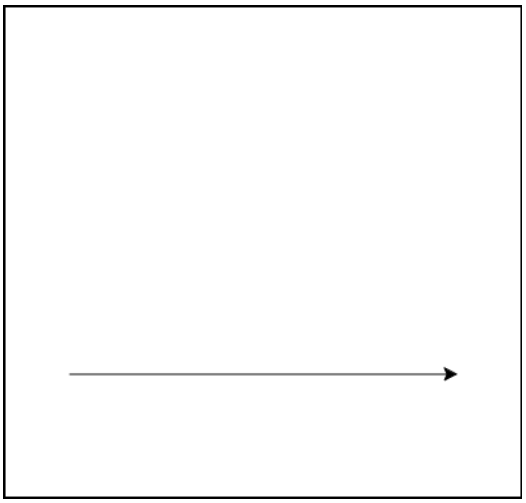
```
def koch(drawer, size, levels):
    if levels == 0:
        drawer.forward(size)
    else:
        koch(drawer, size/3.0, levels-1)
        drawer.left(60)
        koch(drawer, size/3.0, levels-1)
        drawer.right(120)
        koch(drawer, size/3.0, levels-1)
        drawer.left(60)
        koch(drawer, size/3.0, levels-1)
```

That's the entire recursive script, and all that's needed to draw Koch curves at whatever level of detail you want. There are a few things you really need to understand about this code which are not obvious at first glance - make sure you understand *why* each of these is important before you try writing your own fractal drawing code!

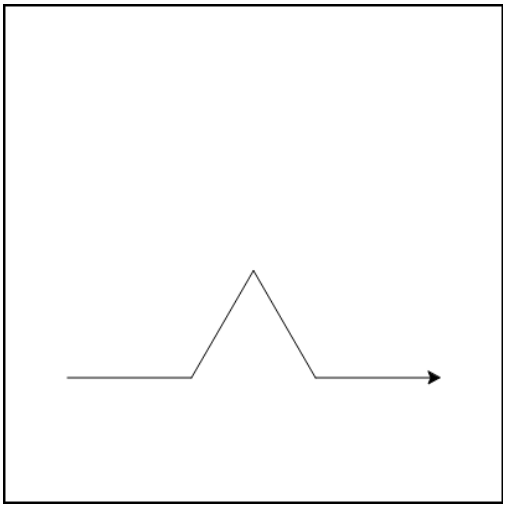
- All direction adjustments are relative adjustments and not absolute - in other words, in recursive code, always use "`turn`" function and never the "`setheading`" function.
- The size used for subproblems (`size/3.0`) depends on the fact that we're turning 60 degrees. If you had a different "seed shape" or even just a different angle in this shape, you'd need to solve a little trigonometry problem to get this right.
- The sequence of turns should always leave the drawing sprite with the same direction that it started in - in other words, all turns should cancel out: in this case we have two counterclockwise turns of 60 degrees, which is exactly balanced by a clockwise turn of 120 degrees.

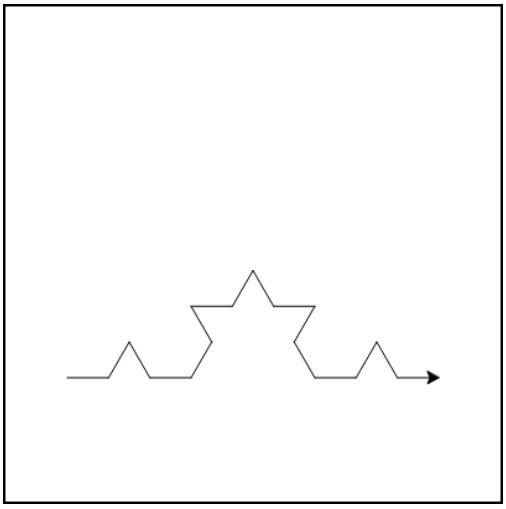
You also need a driver program: a program that starts the drawing by calling the recursive function with the appropriate parameters. The program below shows the driver program that draws a level 0 Koch curve (remember, this is just a straight line) - it has to set the starting position and orientation, make sure the pen is down, and then uses the function we just defined to make the drawing. The output is shown to the right.

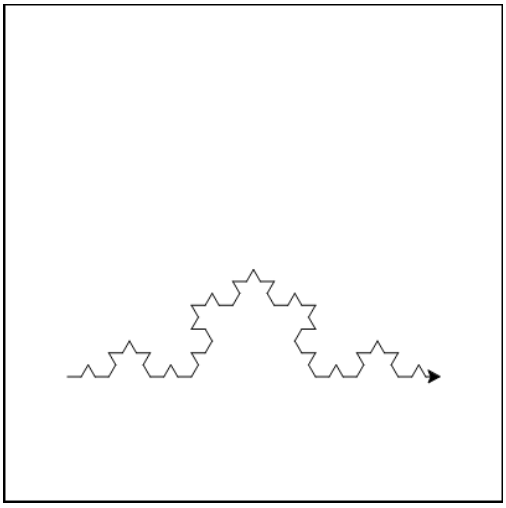
```
tippi = turtle.Turtle()
tippi.speed(0)
tippi.penup()
tippi.goto(-150,-100)
tippi.pendown()

koch(tippi, 300, 0)
```

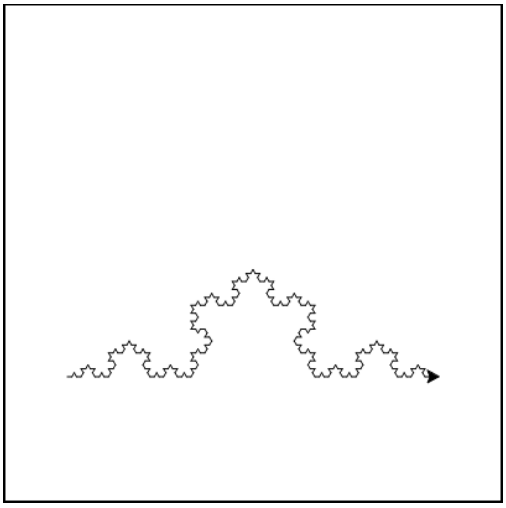


A level 0 Koch curve is very boring, of course. Here's what it looks like if we change the level parameter to 1, 2, 3, and 4 (you should definitely try this for yourself!):









Isn't that cool? Just a little bit of code can draw these amazingly intricate, symmetric, and infinitely detailed pictures!

Now for your assignment: You are to create Python programs to draw two of the fractal designs described in the web lesson you followed for the previous activity, restricted to the ones listed below. I think it would be great if you chose one shape from the "fractals in nature" examples and one from the "African fractals" examples, but really any two will be OK. Here are the shapes you can choose from:

Fractals in nature:

- Organic branching structures ("Application 3")
- Queen Anne's Lace ("Applications 3")
- Fern ("Applications 3")
- Da Vinci branching ("Applications 4")
- Dendrite ("Applications 5")
- River basin ("Applications 5")
- Sierpinski carpet ("Applications 6")
- Lungs ("Applications 7")
- Neuron ("Applications 7" - warning, this one is pretty hard)

African fractals:

- Logone-Bini ("Culture 2" - this one is tricky)
- Golden rectangle ("Culture 2")
- Baila village ("Culture 4")
- Nankani ("Culture 6")
- Fulani wedding blanket ("Culture 9")
- Ethiopian cross ("Culture 11")
- Ghanian bull horn ("Culture 12")
- Kitwe community clinic ("Culture 17")