CS151 Fall 2012 Lecture 21

Stephanie R Taylor

October 24, 2012

1 Writing code to generate strings from L-Systems

Recall that an L-system requires an alphabet, a start/base string, and a set of production rules. We apply the rules for some number of iterations to generate a string.

Today we will write code to generate strings in a 1-rule L-system. We store the L-system in a list, with the first element containing the base string and the second element containing the rule. The rule is represented as a list itself, with the left-hand-side symbol in the first element and the right-hand-side (replacement) string in the second element. To keep the code as readable as possible, we are going to write helper functions that return the base, symbol, and replacement strings from an l-system list.

```
# return the start/base string
def getBase( lsys ):
    return lsys[0]

# return the LHS of the one rule
def getSymbol( lsys ):
    return lsys[1][0]

# return RHS of the one rule
def getReplacement( lsys ):
    return lsys[1][1]
```

To generate a string in the language, we need to know what the L-system is and how many times we need to apply the rules.

```
# Return a string generated by the given L-System, applying
# the rule iterations times.
def generateString( lsys, iterations ):
    ret = getBase( lsys )
    for i in range( iterations ):
        ret = ret.replace( getSymbol( lsys ), getReplacement( lsys ) )
    return ret
```

We can test the code with the L-system to generate a Koch snowflake.

```
if __name__ == '__main__':
    if len(sys.argv) < 2:
        print "USAGE: python lsystem.py <number_of_iterations>"
        print "       to generate a string using the given number"
        print "       of iterations."
        exit()
    koch_snowflake_lsys = ['F—F—F',['F','F+F—F+F']]
    print generateString( koch_snowflake_lsys, int(sys.argv[1]) )
```

2 Interpreting strings with turtle graphics

We can use strings to instruct the turtle to draw shapes, if we interpret each symbol in the string as a turtle drawing command. Let's start with these symbols:

F is forward by a certain distance

\+ is left by an angle

– is right by an angle

Then we can think of a string of symbols as a "shape" for the turtle to draw. If our angle is 90 degrees, then we can draw a square with what string? F+F+F+F.

What information do we need to draw the square? We need the string itself, the angle, and the distance associated with F.

Let's start a new fie (interpreter.py) and put all of our string interpretation code in there. We write a function drawString that takes strings of F, +, and - and interprets them as turtle graphics commands. drawString takes the string and two other pieces of information – the distance associated with any F and the angle associated with any + or -. We loop through the string and do what the character tells us.

```
# interpreter.py
import turtle

def drawString( string, distance, angle ):
  for char in string:
    if char == 'F':
      turtle.forward(distance)
    elif char == '-':
      turtle.right(angle)
    elif char == '+':
      turtle.left(angle)

if __name__ == '__main__':
    turtle.tracer(False)
    drawString( 'F+F+F+F+',100, 90 )
    turtle.update()
    raw_input( 'hold on....')
```

3 Using L-systems to make Fractals

A fractal is a geometric shape in which each piece of the shape is similar to the whole. The term was coined in 1975 by Mandelbrot. Take a look at the Wikipedia pages on L-systems and fractals for more information and for project ideas.

To make more interesting pictures, such as fractals and plants, we use an L-system that will generate the string for us.

We need the functions from Monday's lecture in order to get an L-system to generate a string. So let's open lsystem.py and use that module.

A Koch snowflake can be drawn by the following L-system:

Base F−−F−−F

Rule F → F+F−−F+F

with an angle of 60 degrees.

We update the lsystem.py code to use the Koch snowflake L-system, and have it generate strings using 0, 1, 2, and more iterations. We copy and paste the output into the testing code in lec22 and observe the results.

The string created after 0 iterations is F−−F−−F, which draws an equilateral triangle (see Figure 1).

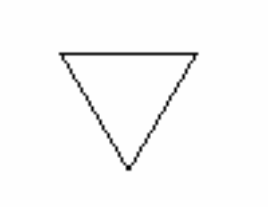

Figure 1: Koch Snowflake with 0 iterations of the production rules

The string created after 1 iteration is F+F−−F+F−−F+F−−F+F−−F+F−−F+F, which draws the Star of David (see Figure 2)

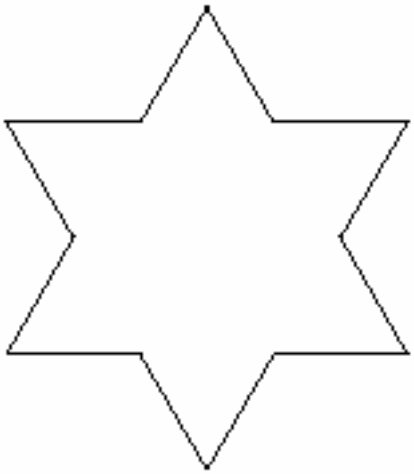

Figure 2: Koch Snowflake with 1 iteration of the production rules

After two iterations, it has ballooned to F+F−−F+F+F+F−−F+F−−F+F−−F+F+F+
F−−F+F−−F+F−−F+F+F+F−−F+F−−F+F−−F+F+F+F−−F+F−−F+F−−F+F+
F+F−−F+F−−F+F−−F+F+F+F−−F+F

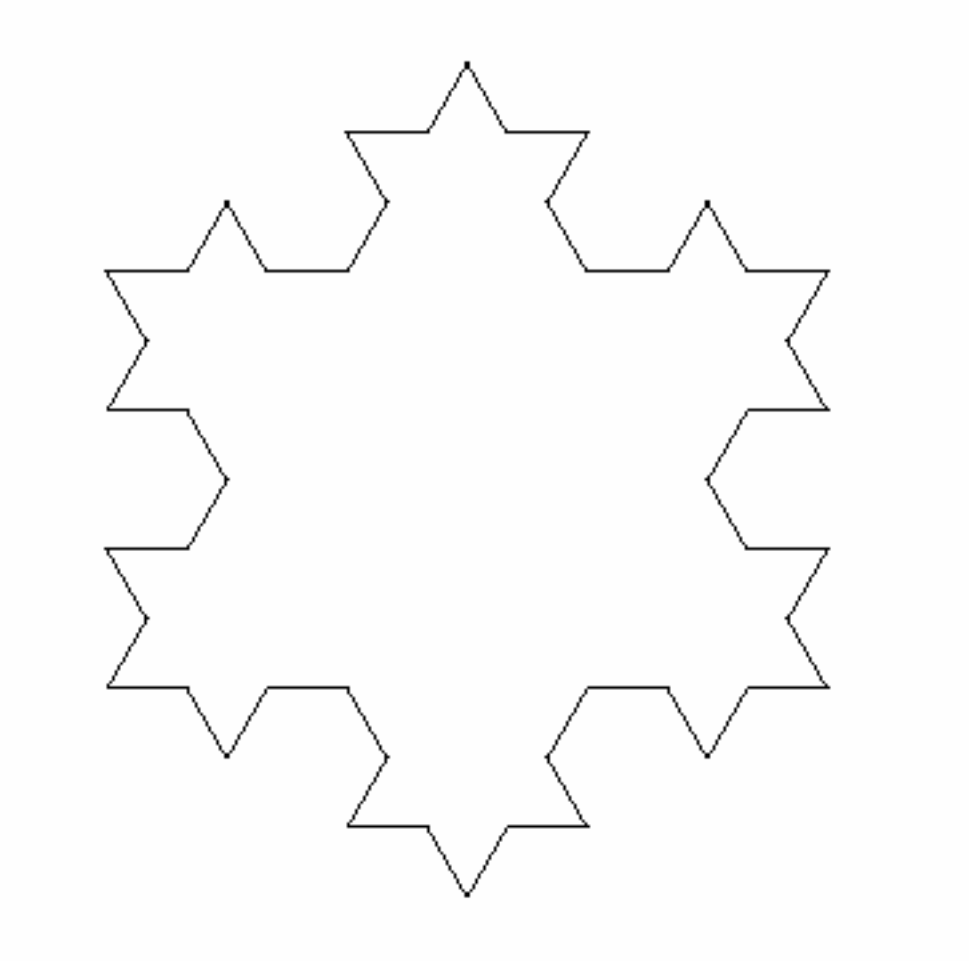

Figure 3: Koch Snowflake with 2 iterations of the production rules

Notice that if you increase the number of iterations, you should decrease the distance.

To draw a snowflake with 5 iterations, I reduce the distance to 1, and it still takes up much of my turtle window. The snowflake is shown in Figure 4. It

is difficult to see the change in size in these notes, but it will become readily apparent when you run the code yourself.

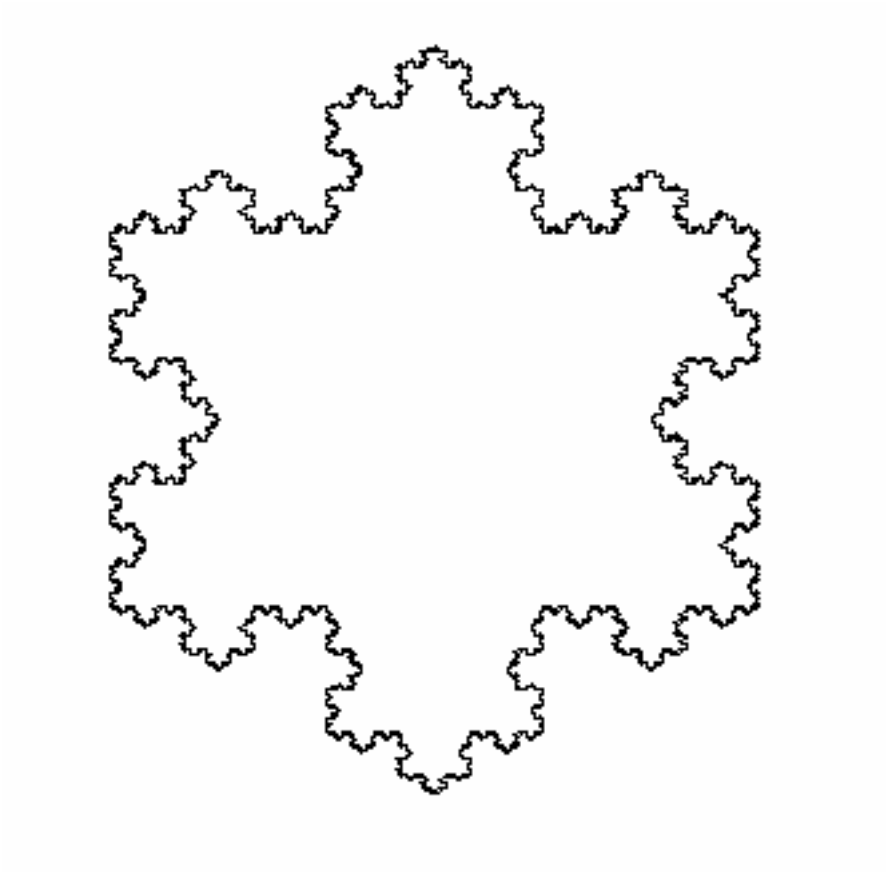

Figure 4: Koch Snowflake with 5 iterations of the production rules