5.8. Sierpinski Triangle

Another fractal that exhibits the property of self-similarity is the Sierpinski triangle. An example is shown in Figure 3. The Sierpinski triangle illustrates a three-way recursive algorithm. The procedure for drawing a Sierpinski triangle by hand is simple. Start with a single large triangle. Divide this large triangle into four new triangles by connecting the midpoint of each side. Ignoring the middle triangle that you just created, apply the same procedure to each of the three corner triangles. Each time you create a new set of triangles, you recursively apply this procedure to the three smaller corner triangles. You can continue to apply this procedure indefinitely if you have a sharp enough pencil. Before you continue reading, you may want to try drawing the Sierpinski triangle yourself, using the method described.

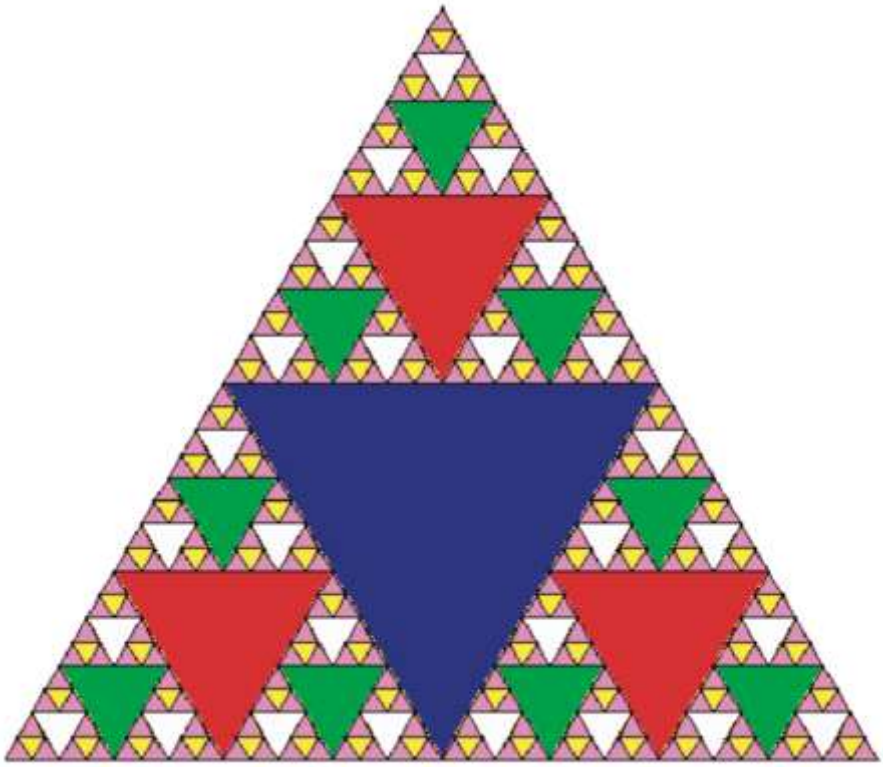


Figure 3: The Sierpinski Triangle

Since we can continue to apply the algorithm indefinitely, what is the base case? We will see that the base case is set arbitrarily as the number of times we want to divide the triangle into pieces. Sometimes we call this number the “degree” of the fractal. Each time we make a recursive call, we subtract 1 from the degree until we reach 0. When we reach a degree of 0, we stop making recursive calls. The code that generated the Sierpinski Triangle in Figure 3 is shown in ActiveCode 1.

Run Load History

```
1 import turtle
2
3 def drawTriangle(points,color,myTurtle):
4     myTurtle.fillcolor(color)
5     myTurtle.up()
6     myTurtle.goto(points[0][0],points[0][1])
7     myTurtle.down()
8     myTurtle.begin_fill()
```

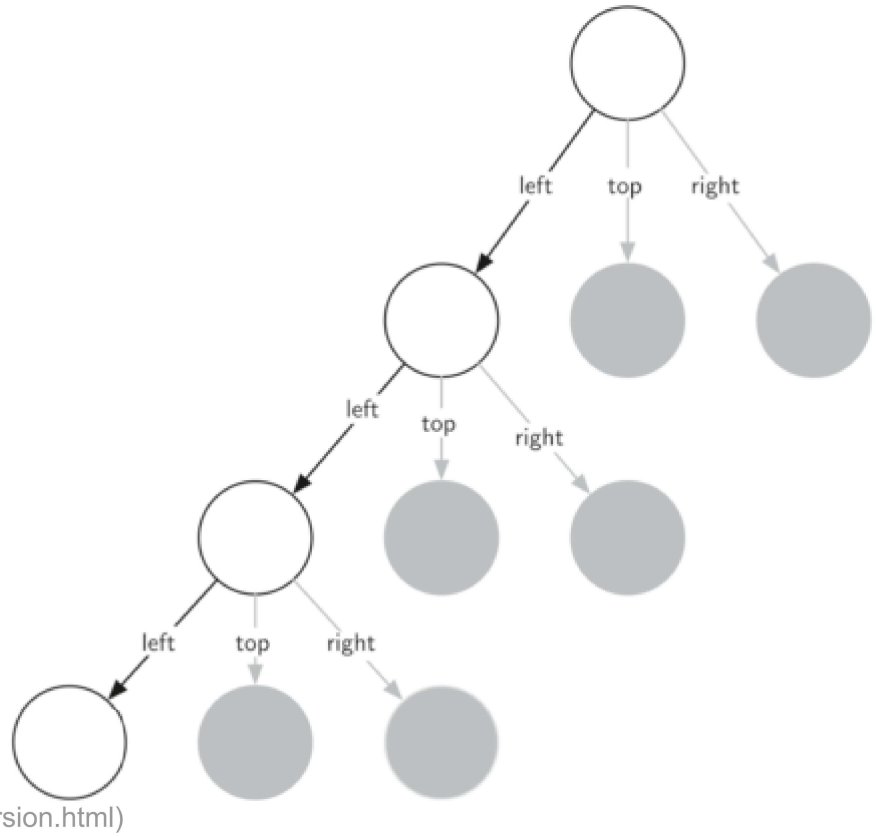
```
    myTurtle.goto(points[1][0],points[1][1])
    myTurtle.goto(points[2][0],points[2][1])
    myTurtle.goto(points[0][0],points[0][1])
    myTurtle.end_fill()

def getMid(p1,p2):
```

Activity: 5.8.1 Drawing a Sierpinski Triangle (lst_st)

The program in ActiveCode 1 follows the ideas outlined above. The first thing `sierpinski` does is draw the outer triangle. Next, there are three recursive calls, one for each of the new corner triangles we get when we connect the midpoints. Once again we make use of the standard turtle module that comes with Python. You can learn all the details of the methods available in the turtle module by using `help('turtle')` from the Python prompt.

Look at the code and think about the order in which the triangles will be drawn. While the exact order of the corners depends upon how the initial set is specified, let's assume that the corners are ordered lower left, top, lower right. Because of the way the `sierpinski` function calls itself, `sierpinski` works its way to the smallest allowed triangle in the lower-left corner, and then begins to fill out the rest of the triangles working back. Then it fills in the triangles in the top corner by working toward the smallest, topmost triangle. Finally, it fills in the lower-right corner, working its way toward the smallest triangle in the lower right.

Sometimes it is helpful to think of a recursive algorithm in terms of a diagram of function calls. Figure 4 shows that the recursive calls are always made going to the left. The active functions are outlined in black, and the inactive function calls are in gray. The farther you go toward the bottom of Figure 4, the smaller the triangles. The function finishes drawing one level at a time; once it is finished with the bottom left it moves to the bottom middle, and so on.



(pythondsintro-VisualizingRecursion.html)
(ComplexRecursi

Figure 4: Building a Sierpinski Triangle

The `sierpinski` function relies heavily on the `getMid` function. `getMid` takes as arguments two endpoints and returns the point halfway between them. In addition, ActiveCode 1 has a function that draws a filled triangle using the `begin_fill` and `end_fill` turtle methods.

You have attempted 1 of 2 activities on this page

user not logged in

(pythondsintro-VisualizingRecursion.html)

(ComplexRecursi