CSCI 136 Assignment – Recursive Graphics

February 15th, 2023

Due Date: Thursday, 2/23/2023 11:59PM

Points: 40

Topic: Recursion, Fractals

In this assignment, you will make a program that draws a Sierpinski triangle. You will then develop a program that draws a recursive pattern of your own original design. You will get practice in using recursion and drawing graphics.

After following all instructions below, please submit your Python scripts (`Sierpinski.py and Art.py`) and a writeup file (`Sierpinski.docx`) in Moodle. If you used any other supporting files, submit these too. Please ensure that all files needed to run your program, including StdDraw.py and related helper files are uploaded with your submission. Upload these as individual files – do not zip them. Be sure the script files have header sections that include the file name, your name, credits, a description of the file, the class and other useful information. Make sure your classes and all methods are thoroughly commented.

- `Sierpinski.py`
- `Art.py`
- `Sierpinski.docx`
- `(supporting files including StdDraw.py)`

Grading:

This assignment is worth 40 points. You will be graded according to the following criteria:

Grade Item	Points
Programs Compiles and Runs	2
Appropriate Comments on All Classes and Methods	2
Sierpinski - filledTriangle Method	4
Sierpinski - sierpinski Recursive Method	8
Sierpinski - main	4
Art - Uses Recursion	5
Art – Original & Dissimilar from H-Tree and Sierpinski	5
Art - Creativity	5
Writeup	5
Total	40

Sierpinski.py

The *Sierpinski triangle* is an example of a fractal pattern like the H-tree pattern shown in our lecture slides on recursion. (Code for the H-tree and variations are included with this lab.) The Polish mathematician Wacław Sierpiński described the pattern in 1915, but it appeared in Italian art since the 13th century. Though the Sierpinski triangle looks complex, it can be generated with a short recursive program.

Your task is to write a program `Sierpinski.py` with a recursive function named `ierpinski()` and a main function that calls the recursive function once and plots the result using standard drawing (StdDraw.py). Think recursively: `sierpinski()` should draw one filled equilateral triangle (pointed downwards) within a larger triangle and then call itself recursively 3 times. There must be an appropriate stopping condition. When writing your program, use a modular design: include a (non-recursive) function `filledTriangle()` that draws a filled equilateral triangle of a specified size at a specified location.

Your program shall take *one* integer command-line argument *N* to control the depth of the recursion. All of the drawing should fit inside the initial equilateral triangle which has endpoints (0, 0), (1, 0), and (1/2, √3/2). **Do NOT change the scale of the drawing window.** First, make sure that your program draws a single filled equilateral triangle when *N* equals 1. Then, check that it draws four filled equilateral triangles when *N* equals 2. Your program should be nearly or completely functional when you get to this point.

Your program `Sierpinski.py` must be organized as a collection of functions with the following specification:

```
---------------------------------------------------------------------------

# Draws a shaded equilateral triangle determined by
# vertex (x, y) and side length s.
filledTriangle(float x, float y, float s)

# Draws one triangle determined by vertex (x, y) and side length s;
# recursively calls itself three times to generate the next (smaller) order
# triangles above, left and right of current triangle
sierpinski(int n, float x, float y, float s)

# Reads depth of recursion N as a command-line argument;
# draws gray outline triangle with endpoints (0, 0), (1, 0), and (1/2, √3/2);
# generates an N-order Sierpinski triangle inside the outline
__main__()
```

Examples

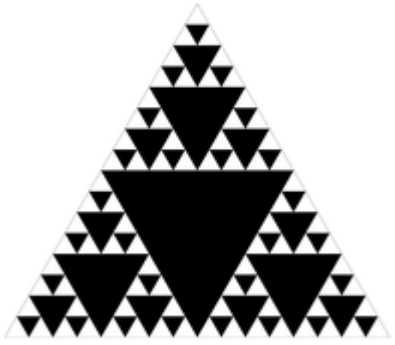
Following are the desired Sierpinski triangles for different values of *N*.

```
% python Sierpinski.py 1
```

```
% python Sierpinski.py 2
```

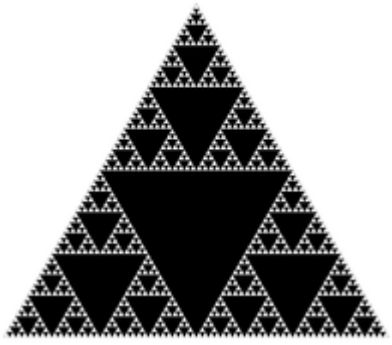
```
% python Sierpinski.py 3
```

```
% python Sierpinski.py 4
```



```
% python Sierpinski.py 5
```

```
% python Sierpinski.py 6
```



Art.py

In this part you will design and create your own fractal. You will write a program `Art.py` that takes one integer command-line argument N (to control the depth of recursion) and produces a fractal pattern of your own choosing. It should work and stay within the drawing window for values 1 through 7. You may choose a geometric pattern like, but not too similar to *Htree* or *Sierpinski*, or a completely different pattern. Your design should not be something that would be easy to generate without recursion. Originality and creativity in the design will be a factor in your grade.

Your `Art.py` program must take one integer command-line argument N (expect it to be between 1 and 7).

Sierpinski.docx (writeup)

Using no more than two pages:

- Include images of the drawings from your Sierpinski.py and Art.py.
- Write up how you generated Art.py, and how you came up with the idea in a short report.

Helpful hints:

Work incrementally. Build and test incrementally, starting with easier aspects. Add functionality incrementally, testing with each change.

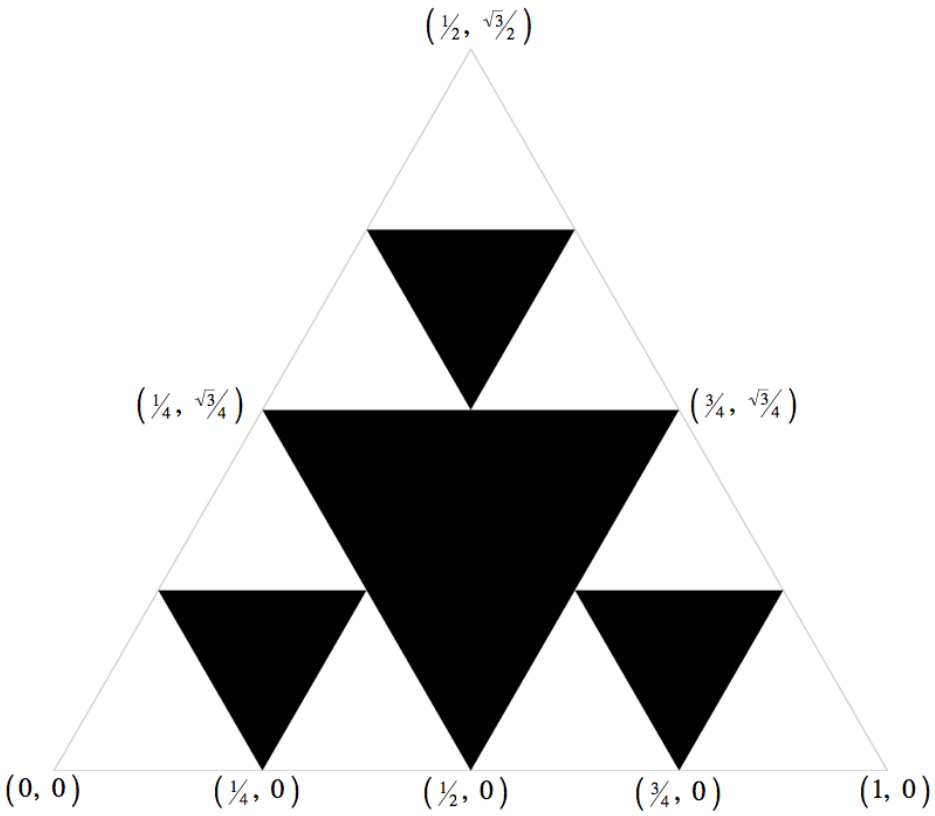
No test script is provided for this lab. You should test thoroughly against the given requirements.

Test and test again. Test thoroughly. Have someone else try to use your program and observe their usage. Fix accordingly.

The Pythagorean Theorem:

You may find the Pythagorean theorem helpful: $a^2 + b^2 = c^2$. The sum of the squares of the two side lengths of a right triangle, a and b, equals the square of the length of the hypotenuse, c. Given the vertices of the initial triangle, which the Pythagorean Theorem is helpful in determining, subsequent vertices can be determined directly as midpoints using simple calculations.

Here are the coordinates of the critical endpoints for the first few levels of the Sierpinski triangle.



How do you draw an equilateral triangle?

Use `StdDraw.polygon()` or `StdDraw.filledPolygon()` with appropriate parameters. Both methods take parallel lists of floating-point numbers, the first containing the x-coordinates of the vertices and the second containing the y-coordinates.

Do not use the following:

- `StdDraw.setCanvasSize()`
- `StdDraw.setXscale()`
- `StdDraw.setYscale()`
- `StdDraw.save()`

Can you use a different color than black to fill in the triangles for Sierpinski.py?

No. For this program use black because it contrasts best with the white background. You can use other colors for Art.py

How should you go about doing the artistic part of the assignment?

This part is meant to be creative and fun, but here are some guidelines. A good approach is to first choose a self-referencing (self-similar) pattern as a target output.

Check out the graphics exercises halfway down the page at:

http://introcs.cs.princeton.edu/23recursion

These are from a text on Java, but the concepts translate to Python.

Here are some recursive graphics submissions from a previous class at Princeton:

https://www.cs.princeton.edu/courses/archive/spring14/cos126/art/index.php

Here are some fractals from Wikipedia:

https://en.wikipedia.org/wiki/List_of_fractals_by_Hausdorff_dimension

What will cause lost points on the artistic part?

We will deduct points if your picture is too similar to Htree or Sierpinski. To be "different enough" from those algorithms, you need to change the recursive part of the program. For example, it is *not* sufficient to simply substitute squares for triangles in Sierpinski. You will also lose points if your artwork can be more easily created without recursion. This might be indicated by a *tail-recursive* function, e.g., a recursive function that calls itself as its last action. For example, the recursive factorial method shown in lecture was tail-recursive.

Can you use image files (.gif, .jpg, or .png) in your artistic creation?

Yes. If you do so, be sure to submit the image files you use along with your other files.

Submit Your Work

After following all instructions above, please submit your Python scripts (`Sierpinski.py and Art.py`) and a writeup file (`Sierpinski.docx`) in Moodle. If you used any other supporting files, submit these too. Please ensure that all files needed to run your program, including StdDraw.py and related helper files are uploaded with your submission. Upload these as individual files – do not zip them. Be sure the script files have header sections that include the file name, your name, credits, a description of the file, the class and other useful information. Make sure your classes and all methods are thoroughly commented.

- `Sierpinski.py`
- `Art.py`
- `Sierpinski.docx`
- `(supporting files including StdDraw.py)`

EXTENSIONS

You will not submit the answers to these extensions as part of your assignment submission. You should work through and understand them to learn more and practice. You may be held responsible for the content within, so you really should work through them. With that said, this are outside the scope of what will be submitted for this lab. Do not change your original submittal to address these extensions.

1.

Create an interactive recursive graphics program using Tkinter. Find a way to use input such as the mouse and keyboard to create an exciting and visually engaging user experience.

2.

Fractal dimension. Somewhere in your mathematical training, you should have learned that the dimension of a line segment is one, the dimension of a square is two, and the dimension of a cube is three. There is a more general meaning for dimension.

Formally, we can define the Hausdorff dimension or similarity dimension of a self-similar figure by partitioning the figure into self-similar pieces of smaller size. We define the dimension to be:

log (# self similar pieces) / log (scaling factor in each spatial direction).

$$\frac{log(\#self\ similar\ pieces)}{log(scaling\ factor\ in\ each\ spatial\ direction)}$$

For example, we can decompose the unit square into 4 smaller squares, each of side length 1/2; or we can decompose it into 25 squares, each of side length 1/5. Here, the number of self-similar pieces is 4 (or 25) and the scaling factor is 2 (or 5). Thus, the dimension of a square is 2 since log (4) / log(2) = log (25) / log (5) = 2. We can decompose the unit cube into 8 cubes, each of side length 1/2; or we can decompose it into 125 cubes, each of side length 1/5. Therefore, the dimension of a cube is log(8) / log (2) = log(125) / log(5) = 3.

We can also apply this definition directly to the (set of white points in) Sierpinski triangle. We can decompose the unit Sierpinski triangle into 3 Sierpinski triangles, each of side length 1/2. Thus, the dimension of a Sierpinski triangle is log (3) / log (2) ≈ 1.585. Its dimension is fractional—more than a line segment, but less than a square! With Euclidean geometry, the dimension is always an integer; with fractal geometry, it can be any fraction. Fractals are widely applicable for describing physical objects like the coastline of Great Britain.

What is the fractal dimension of the pattern you generated for the `Art.py` portion of the lab?