CSC 111aD - Fall 2013 - Lab 12
Due 12/6/13,5pm

Instructions

- For this lab you will answer several questions on the lab sheet and submit it by 5pm Friday. No files will need to be uploaded to Sakai.

Part I – The Fibonacci Sequence

The Fibonacci sequence of numbers [0,1,1,2,3,5, …] was first discussed in a book published in 1202. The sequence is related to the golden ratio and often appears in biological settings such as the arrangement of leaves on a stem. The definition of the Fibonacci sequence is simply:

$F_0 = 0$, $F_1 = 1$, …, $F_n = F_{n-1} + F_{n-2}$

That is, the first two Fibonacci numbers are 0 and 1; subsequent values are the sum of the previous two values. $F_2 = F_0 + F_1 = 0 + 1$, etc.

Write a Python function that receives one positive integer as a parameter, I'll call it N. The function should return the Nth Fibonacci number. Write and test your function.

1) What is the 12th Fibonacci number?
2) What is the 24th Fibonacci number?
3) What is the 36th Fibonacci number?

   Did you notice how much longer that computation took?

   What do you think will happen if you try to compute the 48th Fibonacci number?

   Explain.

4) After you've tested your Fibonacci function in Python, copy it here for grading. You will not need to submit it through Sakai, just write it here (or copy and paste).

Part II – Hilbert Curve

The Hilbert curve is a continuous self-similar (fractal) curve widely used in computer science. A Python program for displaying a Hilbert curve can be downloaded from Sakai in order to answer the following questions. [see Resources → Labs → hilbert.py]

Questions:

1) Run the program and note the appearance of the curve. Experiment with the value of EDGELENGTH by making it smaller. What is the visual effect?

2) Note that the second argument to the hilbert() function is initially 5. Experiment with changing 5 to different values. What is the visual effect?

3) Study the hilbert() function. How does the function conform to the three laws for a recursive function to work properly?
   a.

   b.

   c.

Part III – The Koch (Snowflake) Curve

The Koch Snowflake curve is another interesting “fractal” curve often studied in mathematics. You can find many illustrations on the Internet if you’re not familiar with the idea of a Koch curve. In this part of the lab we will guide you to modify the hilbert.py program to have it draw Koch curves.

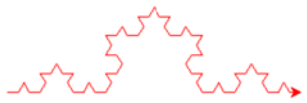
-First, make a copy of the hilbert.py program and name it koch.py.

-Rename the hilbert() function to koch(). The koch() function will have two parameters that I will call *level* and *length*. The *level* parameter has to do with the level of recursion and *length* has to do with the length of line segments the turtle will draw. Here’s the algorithm for the koch() function:

- If level is equal to 0, use the turtle to move forward “length” pixels
- Otherwise,
  - Call the koch() function, passing as arguments, level-1, and length/3
  - Turn the turtle left 60 degrees
  - Call the koch() function, passing as arguments, level-1, and length/3
  - Turn the turtle right by 120 degrees
  - Call the koch() function, passing as arguments, level-1, and length/3
  - Turn the turtle left by 60 degrees

The rest of the program (outside the function) will need two modifications.

1. Have the turtle go to (-200,100) before it starts the drawing
2. Call the koch() function with arguments of 3 and 300

At this point you should be able to run your program and draw a single line Koch curve. You should see something like 

Add the following function to your program:

```
def full_flake( levels, length ):
      for i in range(3):
          koch( levels, length )
          t.right(120)
```

remove the function call to koch() from your main program and replace it with the following: `full_flake( 3, 300)`. Run the program. Experiment with the arguments, trying different values for levels and length.

- Describe the curves your program generates. What is the effect of changing the number of levels and length?