3/26/2015

CS 105
Introduction to Computer Programming using Python
Lindenmayer Systems: Part 2

Instructor:
Joel Castellanos
e-mail: joel@unm.edu
Web: http://cs.unm.edu/~joel/
Office: Farris Engineering Center (FEC) room 319

Spring 2015 Midterm Exam Results

<table>
<thead>
<tr>
<th>0</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>73 Total Students</td>
</tr>
</tbody>
</table>

Mean = 71.23
Median = 74

Anyone who scores better on the final exam, will have his or her final exam grade counted as both the midterm and final.
Lab 7: Lindenmayer Systems: Part 2

Lab 5: Gets (and verifies) input: length, angle, generations, axiom, and two rules.

Lab 7:
1. Reuse lab 5 code to get and verify input.
2. Apply input production rules to the input axiom for the input number of generations.
3. Draw the commands in the final string produced in (2).

Lab 5: Input Format Example #1 (the dragon)

- Axiom = "f"
- Rule 1 = "f = f-h"
- Rule 2 = "h = f + h"
- Angle = 90
- Length = 50
- Generation = 4

<table>
<thead>
<tr>
<th>Generation</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>f-h</td>
</tr>
<tr>
<td>2</td>
<td>f-h - f+h</td>
</tr>
<tr>
<td>3</td>
<td>f-h - f+h - f-h + f+h</td>
</tr>
<tr>
<td>4</td>
<td>f-h - f+h - f-h + f+h - f-h - f+h + f-h + f+h</td>
</tr>
</tbody>
</table>

How quickly does the string grow?
Full Program Overview

1) Get each user input item:
   a) Get input value.
   b) Remove spaces from string values.
   c) Check for errors (it will be easier to check for errors after the spaces are gone).
   d) If error, loop back, if good, get next input...

2) Outer Loop: Loop through number of generations.

3) Inner Loop: Apply production rules to build the next generation from the current generation.

4) End outer loop with final string of final generation.

5) Draw the final generation using turtle graphics.

Details of Algorithm for Production Rules

1) Set a string containing the current generation equal to **axiom**.

2) Outer loop: Loop through each generation.

3) Set temporary variable, **nextGen**, to empty string. This string will be the space where you build the next generation.

4) Traverse the current generation string:
   a) If the current character, c, equals the left side of rule1, then concatenate the right side of rule1 onto the end of **nextGen**.
   b) Else if c equals the left side of rule2, then concatenate the right side of rule2 onto the end of **nextGen**.
   c) Otherwise, c must be a terminal, so concatenate it to the end of **nextGen**.

5) Replace the current generation string with **nextGen**. This finishes the inner loop.
The Sierpiński triangle is a fractal named after Wacław Sierpiński who described it in 1915. It was originally constructed as a curve (as it is with this L-system). It can also be constructed using the “Chaos Game”, or by using an Iterated Function System. The axiom of the Sierpiński triangle is:

**Axiom = "f"**

**Rule1 = "f = h - f - h"**

**Rule2 = "h = f + h + f"**

Length = 2
Angle = 60
Generation = 9

---

The Space-Filling Peano Curve (gen 1-3) has the following axiom:

**Axiom = "f"**

**Rule1 = "f = af + bh + h - af - f f - bh +"**

**Rule2 = "h = - af + bh h + h + af - f - bh"**

Angle = 60
Space-Filling Peano Curve (gen 4-5)

Space-filling curves or Peano curves are curves, first described by Giuseppe Peano (1858 – 1932), whose ranges contain the entire 2-dimensional unit square (or the 3-dimensional unit cube).

The idea of a 1-dimensional object being space filling was found to be highly counterintuitive.

In their limit, the Koch Snowflake, the Sierpiński Triangle, and the Peano Curve all are everywhere continuous with infinite length, and are nowhere differentiable.

Lab 7: Grading Rubric

[A: 4 point]: Draws the correct image given the axiom:
\[ f-h-f+h-f+h-f-h-f+h-f+h-f+h \]
with rules and angle of the Dragon with length=2 and generations = 0.
Note: the above string is generation 4 of the dragon, but by giving it as an axiom, it checks that the drawing part of your program is working without needing to have the production rules working.

[B: 4 point]: Draws the correct image given the axiom, rules, and angle of the Dragon with length=2 and generations = 9.

[C: 4 points]: Draws the correct image given the axiom, rules, and angle of the Sierpiński Triangle with length=1 and generations = 15.

[D: 4 points]: Draws the correct image given the axiom, rules, and angle of the Peano curve with length=1 and generations = 6.

[E: 4 points]: Draws the correct image given an unknown axiom, rules, angle, length and generations.
Extra Credit [+10] : Stack commands

Implement the stack commands ' [' and ' ]'.

- The ' [' command saves the current state of the turtle.
- The ' ]' command restores the turtle to the state saved by the matching ' [ '.
- Both of these commands are terminals.
- These commands can be nested to any level.
  For example:

  \[ +Bf [+Gff [+Rf]] f \]

Extra Credit [+10] : Age Lines

Implement the '!' command.

- The '!' is a terminal.
- The '!' gets older. A newly born '!' draws a line at the current heading of the same length as an 'f' or 'h'.
- The length of each '!' increases by a factor of 1.5 with each generation it is old.
- The color of each '!' must get darker with each generation it is old.
L-System Output: Fern (Generation 1, 2 & 3)

Axiom: \( f \)
Rules:
\[ f=\![+f][-----f]！[+++f]！[--f]！[-f]！[-f]！\]
Initial Angle: 90°
Turn Angle: 8°
Growth: 2.5

Note: The main stems drawn with ! symbols need to show color aging. The tips are drawn with \( f \) and can be any color (except the background color). All of the tips must be the same color.

L-System: Fern – generation 5

Axiom: \( f \)
Rules:
\[ f=\![+f][-----f][++f][--f][-f][+f][-f][-f] \]
Initial Angle: 90°
Turn Angle: 8°
Growth: 2.5
Generations: 5
Extra Credit [+10] : Unlimited Rules

When the user is entering rules, allow him or her to enter as many rules as desired. Then, when building generations, apply those rules. The best way to do this is to store the rules in a list.