Computer Science AP/X — CSCI-140/242
Lindenmayer System — Lab 1

08/08/2024

1 Implementation (75%)[1]

You will **individually develop** a program called `lsystem.py` that implements a string rewriting L-system and draws the graphical representation of the resulting string using the turtle library. For more information about what is an L-system, please refer to the problem-solving document.

1.1 Program Operation

The program will prompt to the user to provide some information via the standard input. You do not have to deal with erroneous input. It is assumed that all inputs are legal, and are all formatted correctly.

When run, the program should execute as follows:

1. Prompt for the axiom. The axiom will be single string of symbols.
2. Prompt for the number of production rules (positive integer value).
3. Prompt for the production rules, one at a time. Every production rule will be a single string of the format $P = S$, P will be a single symbol and S will be a single string of symbols.
4. Prompt for the angle of rotation used by all `left` and `right` turtle commands. (integer value between the range 0-360, both inclusives).
5. Prompt for the initial length of the line segment used by the `forward` turtle command (positive float value).
6. Prompt for the initial turtle's heading (integer value between the range 0-360, both inclusives).
7. Prompt for the number of rewriting steps (the number of times the production rules will be applied, positive integer value).
8. Apply the L-system and print the resulting string to the standard output. **Remember: the axiom is always the first step of the rewriting process.**

1. This lab is inspired by the book Algorithmic beauty of Plants by Przemysław Prusinkiewicz and Aristid Lindenmayer (`http://algorithmicbotany.org/papers/abop/abop.pdf`)

9. Interpret the L-system. As each symbol is interpreted and drawing happens, the corresponding turtle command must be displayed to standard output. For example, if the symbol is `F`, the output printed should be `forward( distance )`.
10. When finished, the program should "idle on" the `mainloop` call until the user closes the drawing window.

1.1.1 Sample Run

Your program is required to generate an output similar to the one below. It must show first the resulting string generated after applying the L-system, and then, it must print all the turtle commands executed in a separate line.

```
Welcome to the L-system drawing generator!
Enter axiom (initial string): Ff
Enter the number of rules: 2
Enter rule #0: F=-F++G-
Enter rule #1: G=+F--G+
Enter angle of rotation: 45
Enter initial line segment's length: 50
Enter initial heading: 0
Enter number of steps: 3
Generating the string ...
Result:
--F++G-+++F--G+-f
Drawing ...
right(45)
right(45)
forward(50.0)
left(45)
left(45)
forward(50.0)
right(45)
left(45)
left(45)
left(45)
forward(50.0)
right(45)
right(45)
forward(50.0)
left(45)
right(45)
pen up
forward(50.0)
pen down
```

Using the sample input from above, the following should be graphically displayed.

2 L-system Alphabet

The program will use the following alphabet for the L-system:

2.1 Drawing symbols

Symbol	Description
`F`, `G`	Both symbols move the turtle forward the given distance with pen down (drawing a line).
`f`	Moves the turtle forward the given distance with pen up (no line drawn).
`+`	Rotates the turtle to the left a given angle without moving.
`-`	Rotates the turtle to the right a given angle without moving.

2.2 Branching symbols

Symbol	Description
`[`	Store the current turtle's state onto the stack.
`]`	Take off the latest turtle's state saved from the stack and set it as the current turtle's state.

In the problem-solving document, we defined the turtle's state as its position and orientation only. For the final lab, the turtle's state is comprised by position, orientation, pen size, pen color, and the length for the line segments.

2.3 Turtle's state symbols

Note: In the table below, `#` in the Description column is a single digit (`0-9`).

Symbol	Argument	Description
`@`	`n`	Multiplies the line segment's length by n, changing the distance moved by any following `forward` command. n is a required float argument with format #.##
`>`	`n`	Changes the turtle's pen color with the color located n steps after the current color in the palette. n is an optional integer argument with format ##. If not indicated, its default value is 1.
`<`	`n`	Changes the turtle's pen color with the color located n steps before the current color in the palette. n is an optional integer argument with format ##. If not indicated, its default value is 1.
`#`	`width`	Set the turtle's pen size thickness to `width`. `width` is an optional integer argument with format ##. If not indicated, its default value is 1.

Note: If the beginning or the end of the palette is reached when using the commands > or <, you will wrap around the other end.

2.4 Turtle Library

You will use a variety of functions from the turtle library to do your drawing:

- `pensize(size)`: Set the pen thickness to the given size. If no argument is given, the current pensize is returned.
- `pencolor(colorstring)`: Set the pen color to `colorstring`, which is a color specification string, such as "black", "blue", etc.
- `xcor(), ycor()`: Return the x and y coordinate respectively.
- `goto(x, y)`: Move the turtle to the given position. You will be required to use this command to restore the turtle's state.
- `heading()`: Returns the turtle's current heading.
- `setheading(angle)`: Set the orientation of the turtle to `angle`.
- `mainloop()`: Halt the program and wait for the user to close the turtle window.

For more information about other turtle functions, please refer to the documentation.

3 Color Palette

To make the drawings more interesting, your program must be able to draw using different colors for the line segments. Turtle's palette offers 256 different colors to choose from, but for this lab, defining a palette with minimum **ten different colors** is sufficient. The colors in the palette is up to your discretion.

The symbols < and > must allow us to move through the palette in both directions and change the turtle's pen color.

4 Samples run

You have been provided with some test files to help you verify the correctness of your implementation. Here is a brief description of what is provided to you:

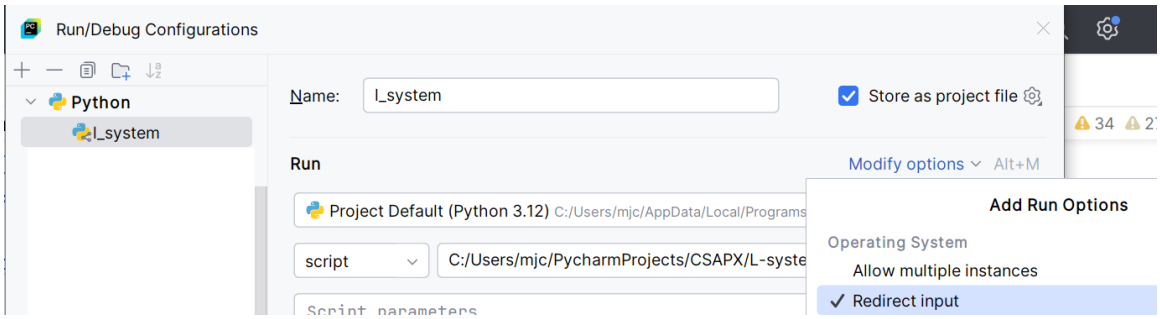
1. **input**: A folder containing different files with the input required to run the program.
2. **output**: A folder containing the expected output when the program run with the **input** files.
3. **images**: A folder containing the expected drawing when the program run with the **input** files.

Notice that we have used different palettes when running the input files. The colors used by the program are part of the output printed in the standard output. You are not expected to match those lines from the output files. The colors used in your palette are up to your discretion. Also, note that, when your program draw the images, they may be off center. This is OK.

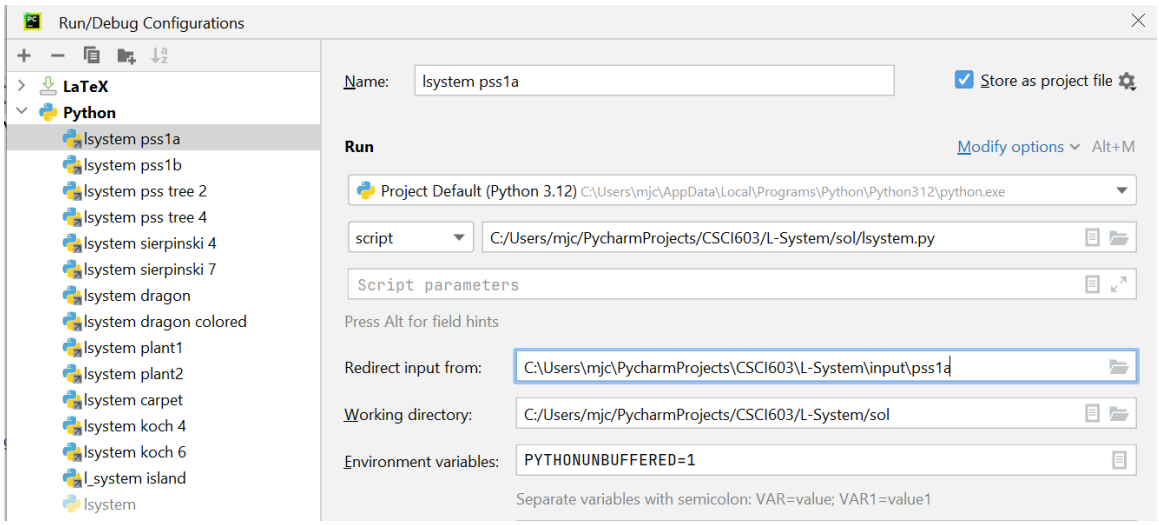
4.1 Simplifying Input

In order to save you the pain of having to cut/paste the user inputs into the console every time you run the program, you can edit the run configuration to automatically redirect the input from a file. To do this, click on the run configuration pull down menu (left of the green run arrow) and select **Edit configurations**.

For the `lsystem` run configuration, click on the **Modify options** pull down menu and enable the option **Redirect input**.



After that, you can specify the full path to any of the **input** files provided in the section **Redirect input from**.



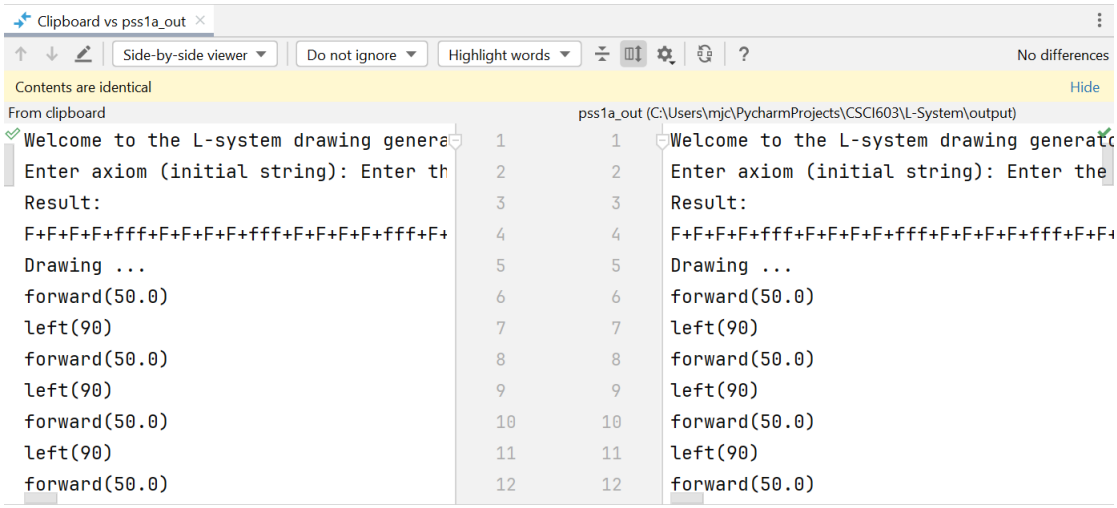
Now when your program executes an `input()` call it will read from the file versus asking you for input.

4.2 Comparing Output

Some of the supplied outputs have a lot of text and can be very confusing to compare your output to. PyCharm provides a utility so you can compare a text file to the clipboard in a visual manner.

1. Open one of the output files in the Pycharm editor window.
2. Run the program with your desired input file.
3. Select all the text in the console and copy it to the clipboard.

4. In the solution output editor window right click and select **Compare with Clipboard**.
5. A new editor window will open highlighting the differences if there is any.



The ultimate goal here would be to make sure the output matches exactly, except for the colors.

5 Implementation details

- You are not allowed to use regular expressions, all the interpretation of the L-system string should be done using `str` operations, e.g. slicing, concatenation, indexing, etc. To check whether a character is a number, use the `str` operation `isdigit`.
- You are not allowed to use the `global` keyword.
- In order to receive full design points, you should be using functions to break down your program. For example, you should have one function that apply the L-system, and a function for evaluating and drawing then graphical representation of the system. Look at your problem solving for more hints about how you can break this down to promote function reuse.
- Your program must be properly documented to receive full style credit. The program should have a main docstring containing your name and a description of the assignment. Each function should have a properly formatted docstring with a description, arguments, return, etc.

6 Grading

The assignment grade is based on these factors:

- Problem Solving: 15%
- In-Lab Activity: 10%

- Functionality: 55%
  - User input: 5%
  - Drawing symbols: 20%
  - Branching symbols: 10%
  - Turtle's state symbols: 20%
- Design: 10% - Your implementation uses functions to promote code reuse.
- Code Style and Documentation: 10%

7 Submission

Go to your project's `src` folder and zip it up. Rename the zip file to `lab1.zip`. Upload this zip to the MyCourses Assignment dropbox by the due date.

- To zip on Windows, right click on the `src` folder and select `Send to -> Compressed (zipped) folder`.
- To zip on MacOS, right click on the `src` folder and select `Compress "src"`.