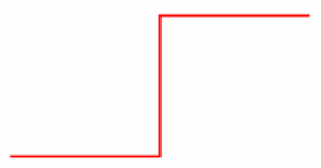
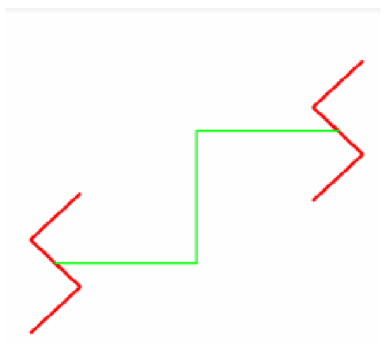
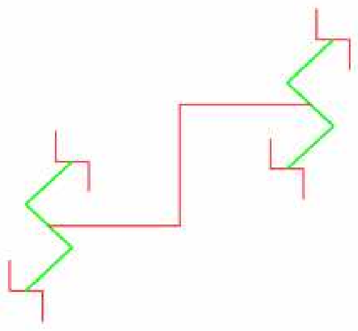
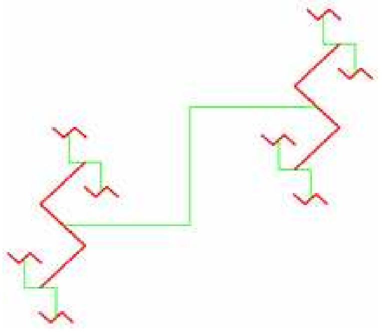
Computer Science I — CSCI-141
ZigZag — Lab 2

01/21/2022

1 Problem

A *zigzag* looks like two treads of a single stair-step. Write functions and compose a program that draws figures like the pictures shown below. When run, the program must prompt for the recursion `depth`, and then draw the figure so that it fits in the standard canvas window. Assume that `depth` will always be non-negative.

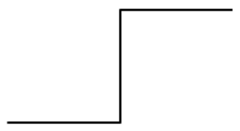
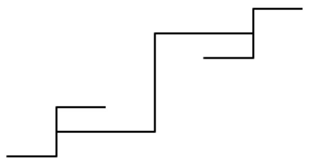
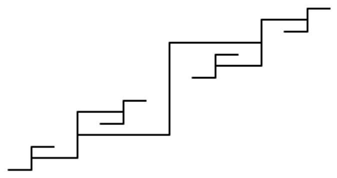
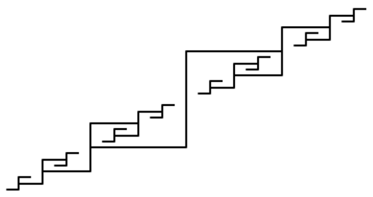
Table 1: Example figures for `depth` values 1-4

2 Problem-Solving Session (15%)

For problem-solving, let's simplify the problem by not tilting the smaller *zigzags* and not changing the drawing color for each level. (You will have to implement tilting and coloring individually in your final solution.)

Table 2: Example figures for `depth` values 1-4 (problem solving version)

Problem-solving involves teamwork with several other students.

1. Write code for a function, `draw_one_zig`, that draws a *zigzag*. The function should take one parameter, `length`, representing the length of a one of the segments of zigzag. Indicate what will be the state of the turtle when the function is called, and the function should move the turtle back to the same state when it completes. Additional helper function would be acceptable.

2. Write code for a non-recursive function, `draw_zigzag1`, that takes one parameter, `length`, and produces the drawing for `depth == 1`. Use the previous function(s).

3. Write code for a non-recursive function, `draw_zigzag2`, that takes one parameter, `length`, and produces the drawing for `depth == 2`. Use the previous function(s).

4. Write code for a non-recursive function, `draw_zigzag3`, that takes one parameter, `length`, and produces the drawing for `depth == 3`. Use the previous function(s).

5. Write code for the recursive drawing function, `draw_zigzag`, that works with any value of `depth >= 0`, and any value of `length >= 0`. It should draw nothing when `depth == 0`.

At the end of problem-solving, deliver your team's work with the items numbered and all team members' names clearly legible.

3 In-lab Session (10%)

After the end of the problem-solving session, begin work independently to complete the following tasks for upload to the postPSS MyCourses dropbox:

You will work on making a program `zigzagBW.py` that will implement the recursive black and white zigzag that you worked on in problem solving.

1. Write a program `zigzagBW.py` that will implement the recursive black and white zigzag that you worked on in problem solving.
2. You only need to include code for the recursive drawing function, `draw_zigzag` (not for non-recursive functions).
3. You should have a `main` function that prompts the user for the depth of recursion. Don't forget to convert that input to an integer.
4. Call your recursive drawing function and provide the initial `depth`, and an initial length of `100`.
5. Keep the window up until the user closes it by using turtle's `done` function.

Do not be afraid to ask for help from the SLI, TA, instructor, or mentoring center if you are having trouble!

For the postPSS, zip your write a `zigzagBW.py` work in progress to a file named **lab02.zip** and upload it to mycourses. You could do this at the end of the in-lab or later before the postPSS Sunday deadline.

4 Implementation (75%)

You will write two programs, the problem-solving version in black-and-white, and the color version.

4.1 PostPSS Version (10%)

Transform the `draw_zigzag` function into a fruitful function that generates black and white *zigzag* figures and returns the length of all the *zig-zag* segments.

The function returns:

- 300 for `length == 100` and `depth == 1`,
- 600 for `length == 100` and `depth == 2`,
- 900 for `length == 100` and `depth == 3`,
- 150 for `length == 50` and `depth == 1`.

You should have a `main` function that prompts the user for the depth of recursion. Don't forget to convert that input to an integer.

The program must draw the figure so that it fits in the standard canvas window, and wait for the user to close the canvas. You will get no credit if not using recursion. Your main function shall print the length of all the *zig-zag* segments:

```
Enter number of levels: 3
Total length of all the zig-zag segments:  900.0
```

Write all functions using the correct style with docstrings. Write *pre-conditions* and *post-conditions* for the function(s) that need this documentation, and provide docstrings for every function in the file, including test functions.

4.2 Color Version (65%)

Implement the program to draw green-and-red *zigzag* figures following the recursive pattern in a file called `zigzag.py`.

You are free to choose whatever color combinations you want.

There are two ways you could go about dealing with switching the drawing color at different depths. The first way would be to use modulo division by 2 (`% 2`) and toggle between setting the two colors. The other way would be to add a Boolean argument to your recursive drawing function that you use to toggle.

4.3 Additional Problem Constraints

1. The turtle may *not teleport (e.g., use* `setpos()`*)* from one location to another. The turtle may lift its pen and move to a new location by a relative amount.
2. Your program must use recursion to draw the figure.
3. For each function, include in its *docstring* comment a description of the **pre-conditions** and **post-conditions** that apply to that function. For example, if a function expects to leave the turtle pen-up and facing North, you should have a line of text in the function's docstring that looks like the one in this definition fragment:
   ```
   def some_function():
       """
       ...
       post-conditions: turtle is pen-up, facing North.
       """
   ```
4. Your program may not use global variables. This prohibition is in place because many students think that a global will help them; in fact, a global variable might cause problems and adversely affect the recursive execution.
5. The figure must be centered in the standard canvas window.
6. The program must prompt the user for the `depth` value. Assume the input value for `depth` is ≥ 0. Error checking of the input is unnecessary.
7. The initial length of one of the segments of the zigzag for `depth == 1` is 100. You don't need to prompt the user for the `length`.
8. The zigzag decreases in size by half at each successive level.

4.4 Notes

For recursive drawings such as this, one way to think of the recursive function is accomplishing two things:

1. drawing some portion of a picture
2. making recursive calls that draw the remaining parts of the picture.

Different people have different valid conceptions of the overall drawing sequence within the recursive function. Here are two such approaches (there may be others):

- A two-stage organization. In stage 1, draw everything needed from this specific instance of the function call. In stage 2, move the turtle to the necessary locations and make all necessary recursive calls. This is the approach we suggested in problem-solving. Within a recursive call, completely draw the current bowtie first, and then secondarily move to all necessary locations to make recursive calls that ultimately draw the remaining bowties.
- A 'call as you go' approach. For some drawings, it may be more efficient or intuitive to intersperse the recursive calls throughout the function. Draw a little bit, make a recursive call, draw some more, make another recursive call. By the time it is done, all the recursive calls have completed, as well as the drawing necessary from this specific function call.

4.5 Grading (75%)

1. 10%: Implementation of the program `zigzagBW.py`.
   - 5% correct drawing;
   - 5% for transformin the `draw_zigzag` function into a fruitful function and printig output in the `main` function.
2. 65%: Implementation of the program to draw green-and-red *zigzag* figures in a file called `zigzag.py`. Again, this program must prompt for the recursion `depth`, and draw the figure so that it fits in the window.
   - 10% correct coloring (different colors are acceptable);
   - 15% tilting and de-tilting;
   - 20% for recursive function designe, including base case (no infinite loops)
   - 10% for style and documentation, including 5% for *pre-conditions* and *post-conditions* for each function;
   - 5% correct `input()` and `int()` usage;
   - 5% for the `main` function.

Failure to follow directions, use only **zip** compression and provide incorrect file names will result in a 10% penalty.

4.6 Submission

Zip the files `zigzagBW.py` and `zigzag.py` (with their appropriate docstring documentation) into a file called `lab02.zip`. Upload your zip file to the appropriate drop box in the MyCourses Assignment area before the due date shown. Be sure to check that the upload completed successfully!!