Algorithm for drawing mosaic patterns

Made by Faycal Kilali for Programming Assignment 2, Oct 14, 2022 for Comp-1631.

Description / Utilization instructions

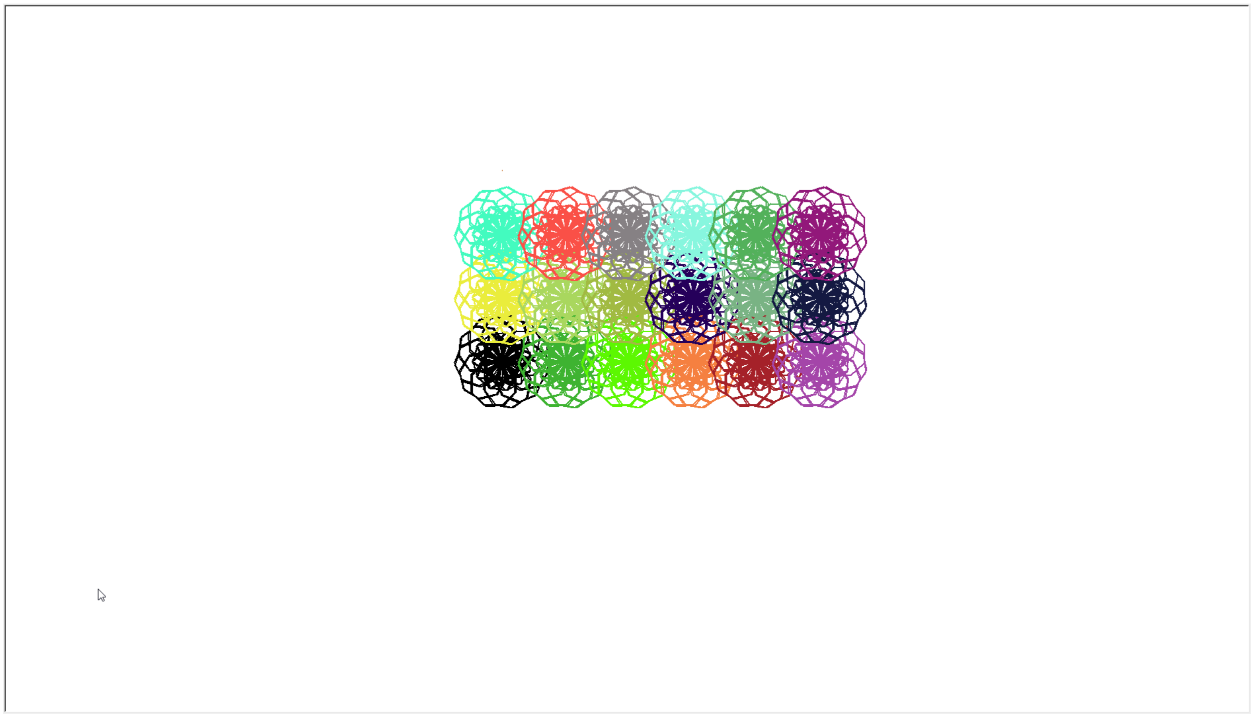
For Scratch: open up A2-Kilali-Faycal.sbc using www.scratch.mit.edu then follow the test suite instructions below.
For Python: open the file in a python IDE, then run the code. Alternatively, invoke python onto the .py file and watch it work! You can invoke it in windows (if your pathing is correct for your Python installation environment) by 'python A2-Kilali-Faycal.py' without the '' marks. For the additional tests, you'll need to edit the scripts as appropriate to the test suite provided.

Test Suite:

The program was tested, and confirmed to be functional with the pre-set values. Here are 3 other tests you may perform.

The intended output is the pattern will be drawn in a row, then go up one row (from the starting row position), then draw the row, then go up one row (from the starting row), then draw the row.

Each row will have 6 repeats (with random colors), and there will be 3 rows in total. In the fashion shown in the introduction of this document



Test 1:

This test was chosen in order to demonstrate that the basic shapes draw properly, in order to compare with the higher number of iterations in the second part. In here, equilateral triangles (shape 1), squares (shape 2), pentagons (shape 3), and octagons (shape 4) will draw a total of 10 times, before the pattern repeats in a different position (as according to the algorithm)

Number of iterations: 10

Length of shape 1 sides: 13

Length of shape 2 sides: 13

Length of shape 3 sides: 13

Length of shape 4 sides: 13

Test 2:
This test was chosen in order to demonstrate that the basic shapes draw properly, even at a high number of iterations. In here, equilateral triangles (shape 1), squares (shape 2), pentagons (shape 3), and octagons (shape 4) will draw a total of 20 times, before the pattern repeats in a different position (as according to the algorithm)

Number of iterations: 20

Length of shape 1 sides: 9
Length of shape 2 sides: 9
Length of shape 3 sides: 9
Length of shape 4 sides: 9

Test 3:

This test was chosen in order to show that the basic shapes still draw properly, even when they vary in side lengths. I*n here, equilateral triangles (shape 1), squares (shape 2), pentagons (shape 3), and octagons (shape 4) will draw a total of 15 times, AND the sizes vary for the shapes. This may not create a perfect mosaic pattern unlike the other tests, but it does demonstrate the functionality of the program in drawing basic shapes in such a pattern. — NOTE: they are in the locus of the geometrical shape, as the octagon gets drawn it drowns them out. Pay close attention to that. They still draw correctly, and that's the goal.*

Number of iterations: 15

Length of shape 1 sides: 16

Length of shape 2 sides: 12
Length of shape 3 sides: 14
Length of shape 4 sides: 16.

Test 4:
Number of iterations: 7
Length of shape 1 sides: 11
Length of shape 2 sides: 11
Length of shape 3 sides: 11
Length of shape 4 sides: 11

This test was chosen for quick-testing changes to the script/program that don't require more thorough diagnosis, the shapes will iterate 7 times, with lengths of 11, until they move on to the next area to iterate through as per the algorithm.

Other variables should remain unchanged, except for the number of sides. Those can be adjusted, they will always result in a *regular polygon* though, unless you have so many sides in order to form a circle (e.g; 360 sides would look like a circle from a distance, but in closer inspection it is not).

Self reflection:

We could provide user inputs for Python and Scratch to provide the user with the option of dictating the side lengths, and even the number of sides. Although, that'll require some restructuring of code as it is not made entirely to be modular in that aspect. That, and well, from what I understand you don't really want me to do that (confirm for me later in feedback).

I also want to provide the user with an option to select the test suite on launch, but that deviates slightly from the algorithm I came up with (as it'll choose the values of the sides and lengths based on the user input) — Would you take marks off me if I include that without adjusting the algorithm to befit it? What if you provide me with an algorithm, how much can I improve on it without you taking marks off me?

Comments:

I'd like to mention that the Turtle package was a bit of a learning curve for the assignment, the differences I noticed were as follows:
In my project of the geometrical patterns in the two languages:

In Python, the drawing is far, far faster than the one that Scratch produces, and I assume this is due to optimization in the different languages.
The more complicated the scripts get in Scratch, the harder they get to debug and analyze. Unlike in Python, I find it far easier to come up with solutions, this assignment is so far one of the few exceptions as per my next paragraph.

I had some serious issues debugging some of the code in Python. Whenever I stopped the process, it gives me an error code that isn’t in fact the culprit (which made me waste 3 hours until I realized I missed implementing one step out of the algorithm)…lesson well learned.
Additionally, the python implementation (when the starting x coordinate is not 0) for some reason the pointer draws a quick line to the starting x coordinate from the origin of (0, 0) before that same line is removed and the normal drawing process begins. Nonetheless, I believe this is a turtle module issue, rather than an algorithm-specific issue. Its super minor, but it would be nice if someone, someday, gives me feedback on why that’s occurring.

Algorithm:

```
define the number of iterations
define the lengths of shape 1 sides
define the number of sides shape 1 is made of
define the lengths of shape 2 sides
define the number of sides shape 2 is made of
define the lengths of shape 3 sides
define the number of sides shape 3 is made of.
define the lengths of shape 4 sides
define the number of sides shape 4 is made of
define x, y coordinates to a feasible point for drawing the mosaic pattern we want
Set our starting coordinate to (x, y) —, and we use that as our starting point.
Set our starting direction to the positive x axis (degree 0 from the x axis). — Note to professor: in Scratch, this is 90 degrees for some reason, and in Python this is already set to 0 anyway.
Repeat the following 3 times
    Repeat the following, 6 amount of times
    Repeat the following, number of iteration times.
        Define a counter to 1
        Repeat the following 4 times
            if the counter is 1, then define number of shape sides = number of shape 1 sides
                define shape side length = shape 1 side length and define number of shape sides = number of shape 1 sides
            elif the counter is 2, then define number of shape sides = number of shape 2 sides
                define shape side length = shape 2 side length and define number of shape sides = number of shape 2 sides
            elif the counter is 3, then define number of shape sides = number of shape 3 sides
                define shape side length = shape 3 side length and define number of shape sides = number of shape 3 sides.
            elif the counter is 4, then define number of shape sides = number of shape 4 sides
                define shape side length = shape 4 side length and define number of shape sides = number of shape 4 sides
            Begin drawing
            Repeat the following amount of times: number of shape sides
                draw a line segment of length of shape side
                rotate left  by 360/num of sides of shape degrees
            rotate left by 360/num of sides of shape degrees.
            increment counter by 1.

Set direction to the positive horizontal axis again (degree 0, in scratch its 90)
Stop drawing
Change color of drawing to a random color
```

```
    Increment horizontal coordinate by length of shape 1 side times 3.5 (and thus so will
    our drawing  coordinate)

Stop drawing & change drawing color to a random color
Define our current x coordinate as our starting x coordinate
Increment vertical coordinate by a distance of length of shape 1 side * 3.5.
Set our drawing position to (x, y).
```