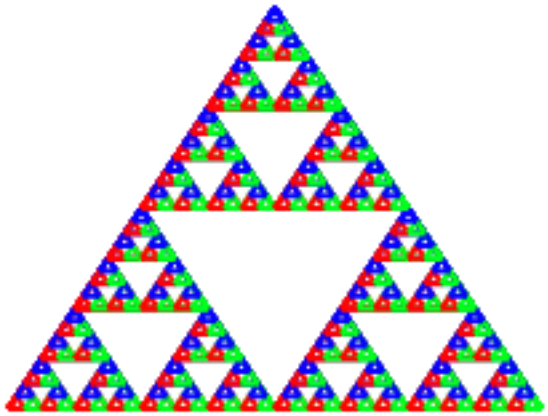
CS 51G Homework Laboratory # 7

Recursion Doodling

Due: by 11 p.m. on **Monday** evening, 3/26.

Objective: To gain experience using recursion.

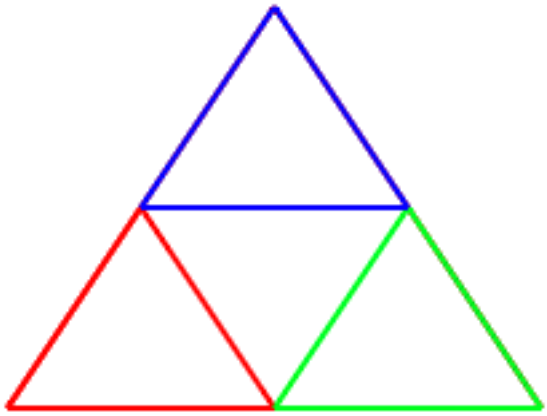
Recursive doodling. This week's assignment has two parts, each of which involves writing a recursive program.



When we doodle, those of us who are less than gifted artistically often resort to making simple geometric patterns on the backs of our notebooks, envelopes, napkins or whatever. One of the all-time favorites is to draw smaller and smaller copies of a single shape inside of another. For example, one can take a shape like a triangle or rectangle and then divide it up into smaller shapes (perhaps by drawing lines between midpoints of adjacent sides) and then divide each of the new regions and so on until all you have is a big blob of ink on the paper.

Fortunately, Grace's drawing resolution is good enough that if we get it to do this kind of doodling, we can produce things quite a bit more interesting than a big blob. For example, repeatedly dividing triangles into smaller triangles can lead to a picture like that shown above.

This image was formed by first drawing a triangle and connecting the midpoints. This breaks the triangle into four smaller triangles (at the top, center, lower-left, and lower-right).



The center triangle is then left untouched, but the top, lower-left, and lower-right triangles are further triangulated in exactly the same way until the length of all of the edges is smaller than 12 pixels long. (In case you are interested, this shape is called Sierpinski's Gasket.)

How to Proceed The starter code is available on the web: http://www.cs.pomona.edu/classes/cs051G/labs/recursion/TriangleStarter.grace is the starter for Part 1, http://www.cs.pomona.edu/classes/cs051G/labs/recursion/StairsStarter.grace is the starter for Part 2.

Because this triangle doodle is going to be a recursive structure, we will provide you with a type for the doodle (called `Movable`). You will create a base object (called `emptyDoodle`), and a recursive class (called `complexDoodle`). The only method that will be needed for triangle doodles is a `moveBy` method, as specified in type `Movable`.

The `emptyDoodle` object represents an empty doodle. Like the empty scribble or empty target, nothing is drawn on the screen, so the constructor and moveBy method are both pretty trivial.

The `complexDoodle` class is more complicated. It will need instance variables or defs for the lines composing the outer triangle as well as instance variables or defs for the three triangle doodles inside.

The class will take parameters for the vertices of the triangle (in the order left, right, and top), its color (i.e., the color for the outer triangle), and the drawing canvas. It should then draw lines of that color between the vertices to form a triangle. It will also construct three more triangle doodles inside this triangle. If any of the edges of the new triangle are long enough (say, with length greater than or equal to 12 pixels), then the constructor should find the midpoints of each of the sides and then create complex triangle doodles in the top, lower-left, and lower-right portions of the triangle (leaving the middle blank). For these complex doodles, the one on the top should be blue, the one on the lower left should be red and the one on the bottom right should be green. If none of the edges are long enough, then just create three empty triangle doodles for the instance variables (as we did with scribbles). The `move` method should move the lines forming the triangle as well as the contained triangle doodles.

We have provided you with a main program that inherits `graphicApplication`. It constructs a new object from class `complexDoodle`. The `onMousePress` and `onMouseDrag` methods use the `moveBy` method to drag the triangle doodle around (even when the mouse is not in the triangle doodle).

To start out, it often a good idea to do a simpler version of the problem. For example, first have the class `complexDoodle` just draw a simple triangle out of lines. Then try drawing only the triangle doodles in the top part of the triangle. Then add the lower right, then finally adding those in the lower left portion.

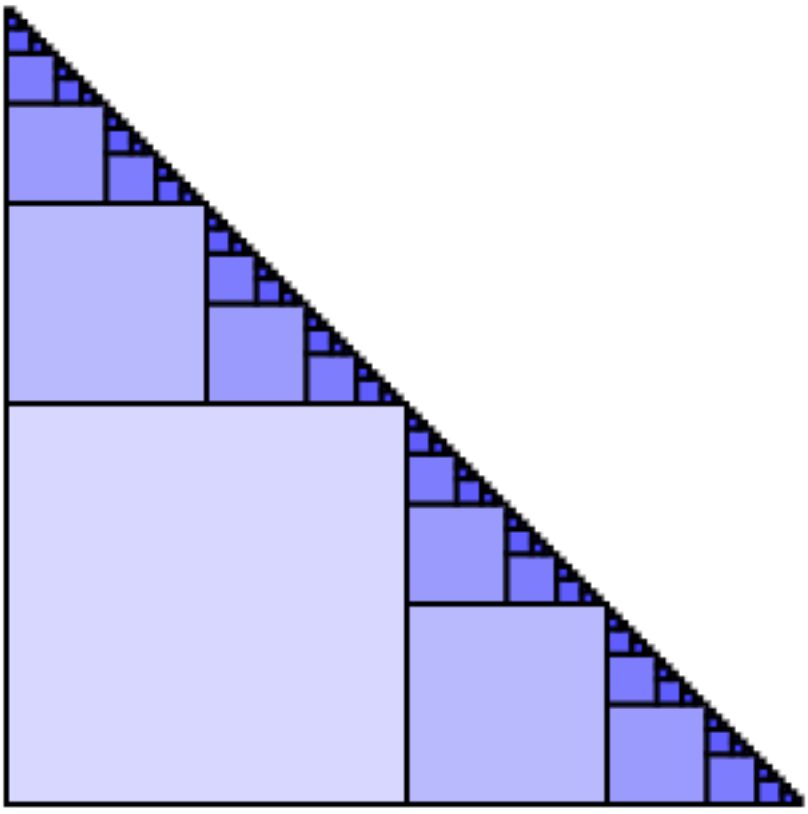
The window should be set to be 400 pixels square for this problem. You will also find the `distanceTo` method defined in the `Point` class useful to determine when to stop the recursion:

```
method distanceTo(Point otherPoint)
```

If you are not careful in writing this program (and the next one), you may get a `StackOverflowError`. This will occur if your program continues to construct new triangle doodles without ever terminating. You can avoid this if you make sure you have written the base class correctly and that the complex doodle invokes its constructor only to create simpler doodles, eventually creating only empty doodles.

Do be aware that as the smaller triangles are drawn on top of larger triangles, the final picture you draw will end up looking more like the one on the first page of this handout.

Part 2: Your second task is to draw the picture given below:



Let's call this variegated stairs (got to love those big words!). A variegated stairs drawing is constructed by first drawing a large square. Then draw variegated stairs half as large on top of the square, and variegated stairs, also half as large, immediately to the right of the square. Stop creating non-empty stairs when the squares are less than 3 pixels on a side.

Each time you draw the stairs you should decrease the red and green components of the color by 30 pixels. That way you will obtain the nice shading as shown above. You can obtain the red, green, and blue components from a color with the methods:

```
red -> Number
green -> Number
blue -> Number
```

As above, we will provide you with a type `Draggable`, and you will need to create an object, `EmptyStairs`, and a class `variegatedStairs`. We have provided the main program. This time that class will only drag the stairs around if the user presses the mouse down on the stairs. Thus you will need to create methods `contains` and `moveBy`. The class header is:

```
class variegatedStairsAt (lowerLeft:Point) size (stairSize: Number)
                    color (stairColor: Color)
                    on (canvas:DrawingCanvas) -> Movable {
```

The first parameter is the lower left hand corner of the large square, while `stairSize` is the length of the size of that square.

Submitting Your Work Labs are due Monday night, 3/26 at 11 p.m.

When your work is complete you should deposit in the appropriate dropoff folder a single folder containing a separate file for each program. Make sure the folder name is of the form `lab7_lastnamefirstname`. Also make sure that your name is included in the comment at the top of each Grace source file.

Before turning in your work, be sure to double check both its logical organization and your style of presentation. Make your code as clear as possible and include appropriate comments describing major sections of code and declarations.

Table 1: Grading Guidelines

Value	Feature
	Readability (3 pts total)
1 pt.	Descriptive comments
1 pt.	Good names, constants
	Semantic quality (3 pts total)
1 pt.	conditionals and loops
1 pt.	Parameters, variables, and scoping
1 pt.	Types
1 pt.	Base and recursive classes
	Correctness (4 pts total)
	Triangles
1/2 pt.	Draws initial triangle correctly
1/2 pt.	Recursive triangle moves correctly
1/2 pt.	Proper sub-division and coloration
1/2 pt.	Recursion stops at correct size
	Stairs
1/2 pt.	Draws initial rectangle correctly
1/2 pt.	Recursive stairway moves correctly
1/2 pt.	Proper sub-division and coloration
1/2 pt.	Recursion stops at correct size