Multiple transformations

Now that you've seen the basics of translation, rotation, and scaling, let's talk about using all of them together, and some of the complexities we brushed over at the beginning.

Order matters

When you do multiple transformations, the order makes a difference. A rotation followed by a translate followed by a scale will not give the same results as a translate followed by a rotate by a scale. Here is an example program that demonstrates that:

```
// yellow square

fill(255, 255, 0);

rect(70, 70, 20, 20);



// red square

pushMatrix();

fill(255, 0, 0);

rotate(30);

translate(70, 70);

scale(2.0);

rect(0, 0, 20, 20);

popMatrix();



// green square

pushMatrix();

fill(218, 232, 193);

translate(70, 70);

rotate(30);

scale(2.0);

rect(0, 0, 20, 20);

popMatrix();
```

Which order you use depends on what your desired effect is. Just keep in mind that you're moving the graph paper, not the object itself, and you should find an order that works for you.

The transformation matrix

Every time you do a rotation, translation, or scaling, the information required to do the transformation is accumulated into a table of numbers. This table, or matrix has only a few rows and columns, yet, through the miracle of mathematics, it contains all the information needed to do any series of transformations. And that's why the `pushMatrix()` and `popMatrix()` have that word in their name.

What about the push and pop part of the names? These come from a computer concept known as a stack, which works like a spring-loaded tray dispenser in a cafeteria. When someone returns a tray to the stack, its weight pushes the platform down. When someone needs a tray, he takes it from the top of the stack, and the remaining trays pop up a little bit.

In a similar manner, `pushMatrix()` puts the current status of the coordinate system at the top of a memory area, and `popMatrix()` pulls that status back out. The preceding example used `pushMatrix()` and `popMatrix()` to make sure that the coordinate system was "clean" before each part of the drawing. In all of the other examples, the calls to those two functions weren't really necessary because there were no subsequent transformations, but it doesn't hurt anything to save and restore the grid status. As a best practice, always use those functions when you're doing any transformations.

There is also a `resetMatrix()` function that resets the matrix back to its very original state (the "identity matrix"), but the push and pop functions are nearly always the better approach.