Translation, Rotation and Scale

We have moved figures by altering the value of the variables that store the (x, y) anchor point of the figure we want to move. This is a very useful way to move figures but there is another… beware… the event horizon is closing…

What if we coded a complex figure with magic number (never…) and now we want/have to move it? And what if we do not have time to recode it with variables and expressions?

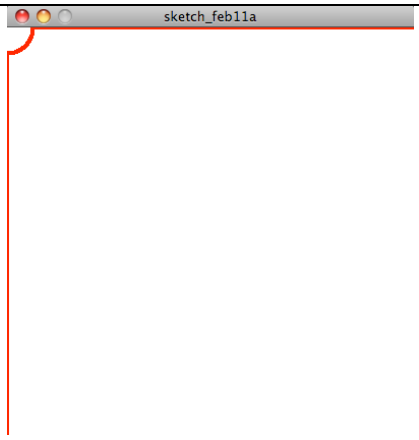
If we cannot alter the variables, Processing allows us to move the origin. We can move the (0,0) point which has a default location of the upper left corner to anyplace we want to.

```
size( 400, 400);
background( 255 );
stroke( 255, 0, 0 );
strokeWeight( 3 );

// vertical line
line( 0, 0, 0, height );

// horizontal line
line( 0, 0, width, 0 );

ellipse( 0, 0, 50, 50 );
```



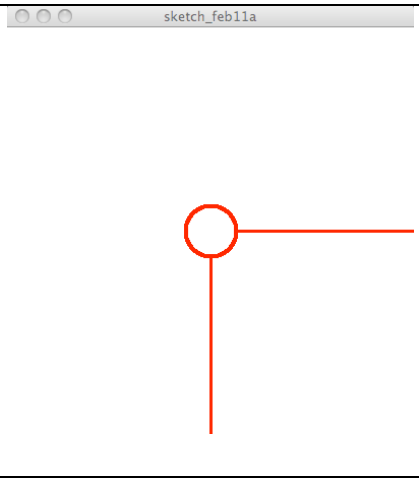
The function that moves the origin this is the function translate( ). The first argument it the amount of x shift in pixels and the second argument is the amount of y shift in pixels.

```
size( 400, 400);
background( 255 );
stroke( 255, 0, 0 );
strokeWeight( 3 );

translate( 200, 200 );
// vertical line
line( 0, 0, 0, height );

// horizontal line
line( 0, 0, width, 0 );

ellipse( 0, 0, 50, 50 );
```



Once we translate, everything is moved to the left or right and up or down the distance dictated by the arguments. This translation affects only figures drawn after the translation and only until the frame is finished being rendered. Every

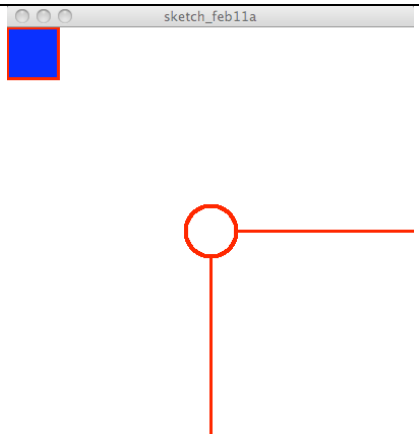
new repetition of the draw( ) function that generates the frame begins with the anchor point in the upper left corner.

If you want to temporarily translate to a new position and then return to the previous coordinate values, you can get a temporary shift but putting the translate and the desired drawing code inside two functions that mark the translate as temporary. These functions are pushMatrix( ) and popMatrix().

In this example the ellipse is drawn within the pushMatrix/popMatrix and the rect is drawn after .

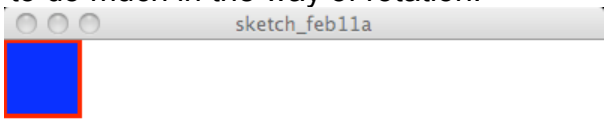
```
size( 400, 400);
background( 255 );
stroke( 255, 0, 0 );
strokeWeight( 3 );

pushMatrix( );
    translate( 200, 200 );
    line( 0, 0, 0, height );
    line( 0, 0, width, 0 );
    ellipse( 0, 0, 50, 50 );
popMatrix( );
fill( 0, 0, 255 );
rect( 0, 0, 50, 50 );
```



Rotation

We can also rotate figures. Unless we want to resort to trig (sin and cos) [1] we have not been able to do much in the way of rotation.



[1] We will resort to this a bit later in the term…

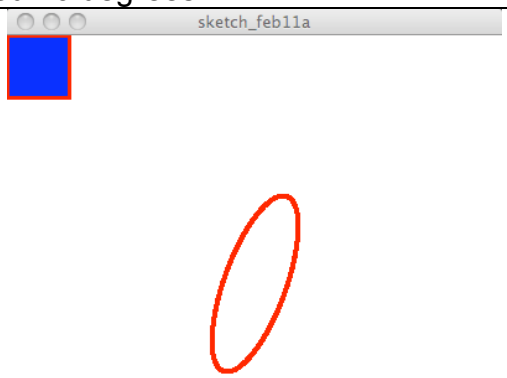
Even with trig we could not have drawn this: without a major exercise in curveVertex( ) experimentation.

This figure was drawn with this code:

```
size( 400, 400);
background( 255 );
stroke( 255, 0, 0 );
strokeWeight( 3 );
noFill( );
pushMatrix( );
    translate( 200, 200 );
    rotate( PI/3 );
    ellipse( 0, 0, 50,150 );
popMatrix( );
fill( 0, 0, 255 );
rect( 0, 0, 50, 50 );
```

The rotate function takes a single argument – a value in radians. This is fine if the amount is easily defined with an expression using PI (the Processing constant for 3.14159).
However, what if we are using the frameCount variable or some other variable to control the rotation? You can do it with PI or you can let Processing do it for you. Processing has a function named radians( ) that takes a single argument (representing an increment of rotation of rotation in degrees). The function radians( ) takes the argument and computes and returns the corresponding radian value. This ellipse has been rotated 20 degrees.

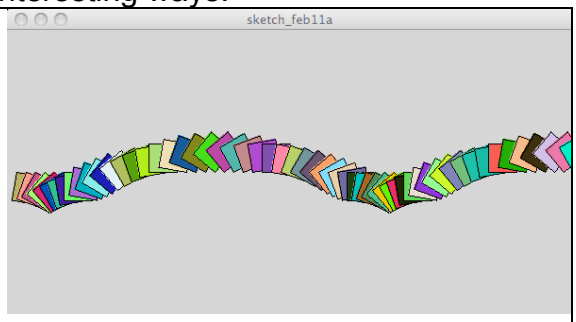
```
size( 400, 400);
background( 255 );
stroke( 255, 0, 0 );
strokeWeight( 3 );
noFill( );
pushMatrix( );
    translate( 200, 200 );
    rotate( radians( 20 ) );
    ellipse( 0, 0, 50,150 );
popMatrix( );
fill( 0, 0, 255 );
rect( 0, 0, 50, 50 );
```



We can mix translations and rotations in interesting ways.

```
void setup ( )
{
   size( 600, 300 );
}

void draw( )
{
 fill(random(255),
       random(255),
        random(255) );
 translate
     ( frameCount*10, 150 );
 rotate
   (radians(frameCount*10));
 rect( 0, 0, 30, 30 );
}
```



Here is one more tool for your toolbox…

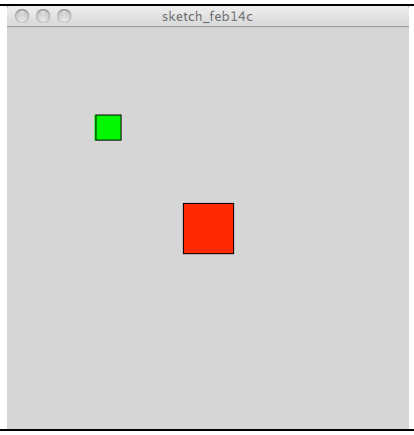
Scale

We can resize figures without altering the argument value. The function scale( ) can reduce and enlarge figures. It can be a bit tricky to use but it might be worth exploring. Take a look at this example:

```
size( 400, 400 );

rectMode( CENTER );
fill( 255, 0, 0 );  // red
rect( 200, 200, 50, 50 );

scale( .5 );
fill( 0, 255, 0 );  // green
rect( 200, 200, 50, 50 );
```



- Both rectangles are drawn 200 pixels over and down from the upper left corner.
- Both rectangles are drawn with edges of 50 pixels.

The green rectangle is drawn after the function scale( )is called. The argument is .5 and the result is that everything is reduced by 50%. Everything drawn after scale( )is called is reduced by 50%.

The change caused by calling scale( ) is in effect only during the remainder of the current frame. Once the rendering of the current frame is finished, the scale of the drawing returns to its default value which would be 1.

And last today but not least (new – no code for this one)

Temporary styles that can be "erased" when you are done.

There are two functions in Processing that allow you to temporarily alter the various values for the color of the stroke and fill and for the strokeWeight or the various modes to draw one thing and then throw them away so everything reverts to the previous settings.

These functions are
pushStyle( );

popStyle( );

The terms push and pop have a long history in programming and revere to the idea of pushing or adding a new item to a stack of stuff – like putting a new sheet of paper on an existing stack. Pop refers to removing the top item on the stack like removing the top sheet of paper on the stack.

Processing keeps what we can think of as a "style sheet" that has all of the values that Processing must use to draw figures. The idea of pushStyle( ) is that it puts a temporary copy of this style sheet on top of the stack. When your code alters the fill or the rectMode, it changes the new copy of the style sheet that is on top of the stack but not the original style sheet that is second from the top in the stack.

When you are finished drawing, you can throw away this temporary style sheet and return the various values to their previous settings by calling popStyle( ).

Check the API for pushStyle( ) to see what can be altered temporarily.

Review of the longevity of function calls that change the drawing environment:

Change extends beyond current frame until changed.	Change extends only until the end of the current frame.
fill( ), noFill( ), stroke( ), noStroke( ), strokWeight( ), rectMode( ), ellipseMode( ), imageMode( ), textSize( ), textAlign( )	translate( ), rotate( ), scale( )

Beware, there may others to add to either list.

Is any of this stuff on the test?

Nope!
No Way!
Never!
Not in Jim's Lifetime!

This is for you to use as you see fit. It is not testable in any way, shape or form.

Seriously…