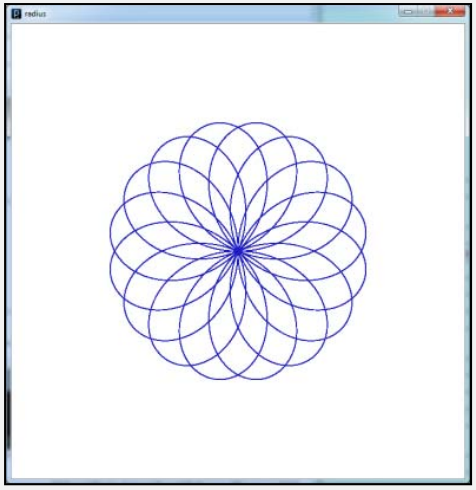
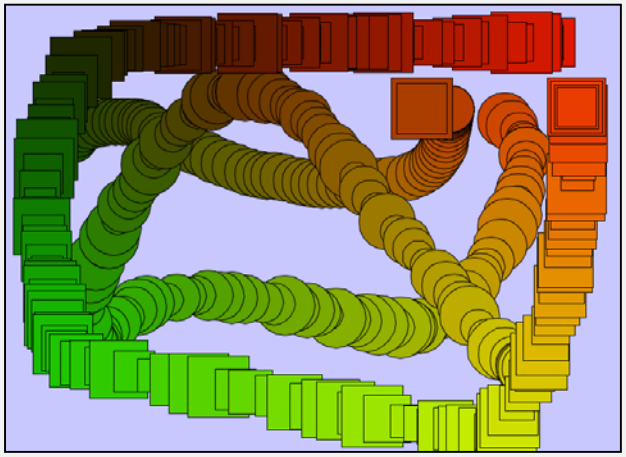
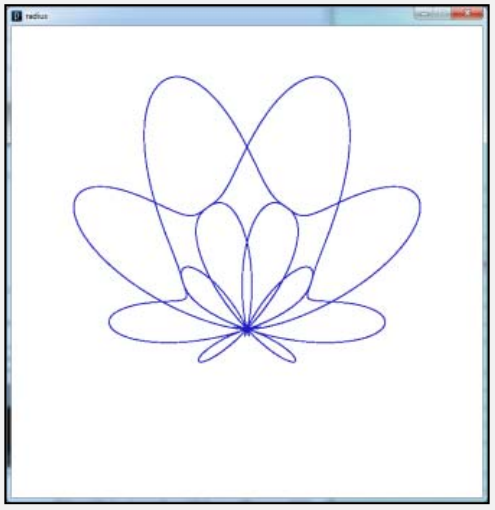
Variables and For-loops







Oregon State University

Mike Bailey

mjb@cs.oregonstate.edu

Variables – using symbols instead of just numbers

Variables are the process of replacing concrete values with symbols in order to generalize a computation to work in more than one situation

```
void
draw(  )
{
  stroke( 0, 0, 0 );
  fill( 255, 50, 50  );
  int x = 100;
  int y = 200;
  rect( x,  y,    150,  50 );
}
```

"int" stands for "integer", a whole number with no decimal digits, e.g., 3

"float" designates a number that can have decimal digits, e.g., 3.14

Variables – using symbols instead of just numbers

We can use variables to capture ***relationships***

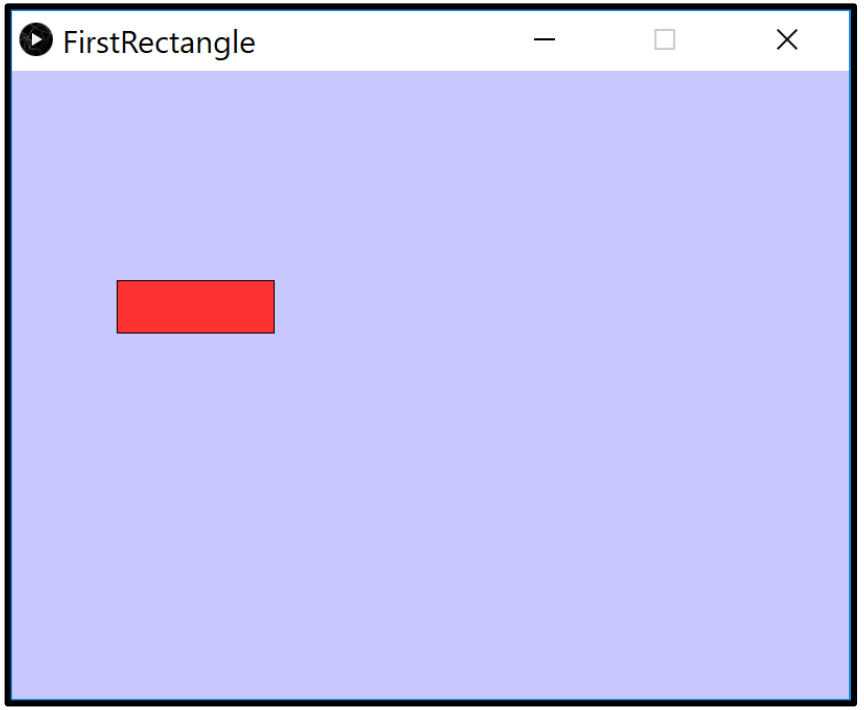
```
void
draw(  )
{
  stroke( 0, 0, 0 );
  fill( 255, 50, 50  );
  int x = 100;
  int y = 2 * x;
  rect( x,  y,    150,  50 );
}
```

Arthmetic operations in programming are:

+	**Addition**
-	**Subtraction**
*****	**Multiplication**
/	**Division**
( )	**Grouping**

Drawing One Rectangle is Pretty Straightforward

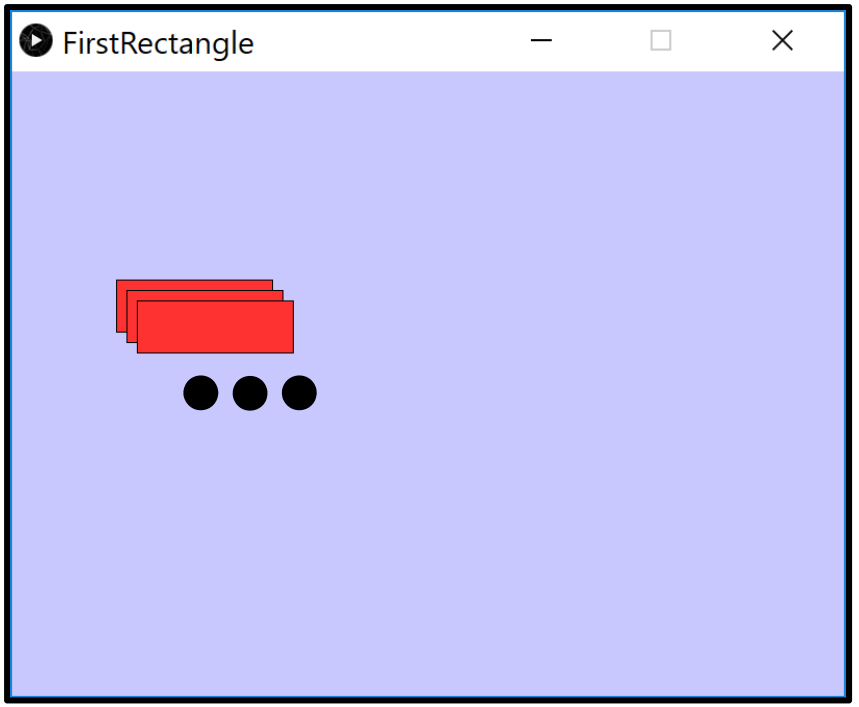
```
rect( 100,  200,    150,  50 );
```



But, This Gets Awfully Boring if You Want to Draw 100 Rectangles!


```
rect( 100,  200,    150,  50 );
rect( 110,  210,    150,  50 );
rect( 120,  220,    150,  50 );
```

●●●



For-loops to the Rescue!

Repeating a code pattern is a recurring theme in programming.

```
void
draw(  )
{
  stroke( 0, 0, 0 );
  fill( 255, 50, 50  );
  for( int x = 0 ; x < 400 ; x = x + 10 )
  {
    int y = x;
    rect( x, y,  150, 50 );
  }
}
```

This line is called a "for-loop". It is very handy for repeating patterns of code. It expresses those patterns as ***relationships***.

The for-loop executes the commands in the curly braces a bunch of times. Its use looks like this:

Keep looping as long as this equation is true

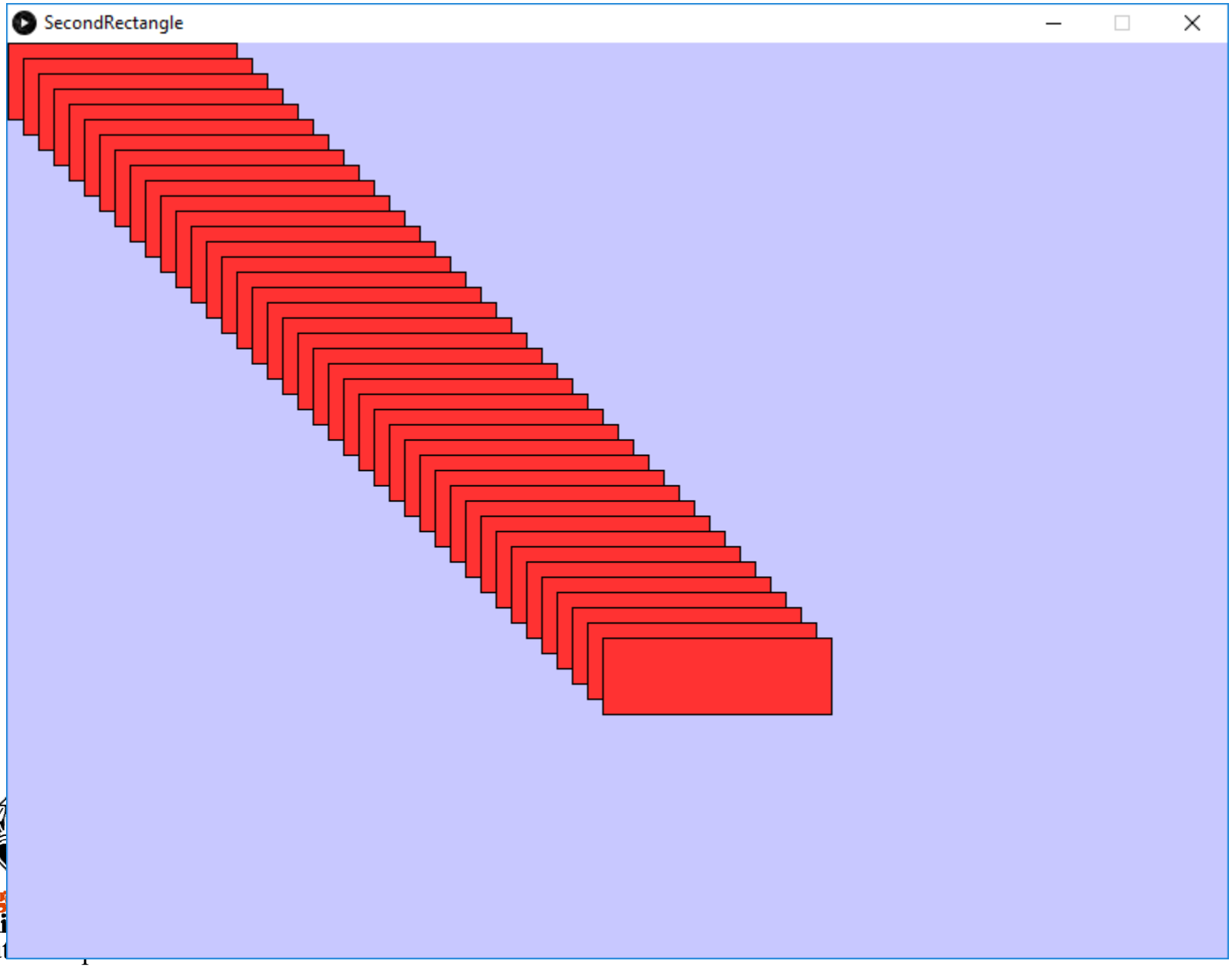
for(□ ; □ ; □)

Do this equation once at the start

Do this at the end of one loop, but before the start of the next one

```
for( int x = 0 ; x < 400 ; x = x + 10 )
```

For-loops to the Rescue!



More Sophisticated Relationships:

The *map( )* function

This function takes an input value, the range of values it lives between, and the range of output values. It returns the output value that corresponds to the input value.

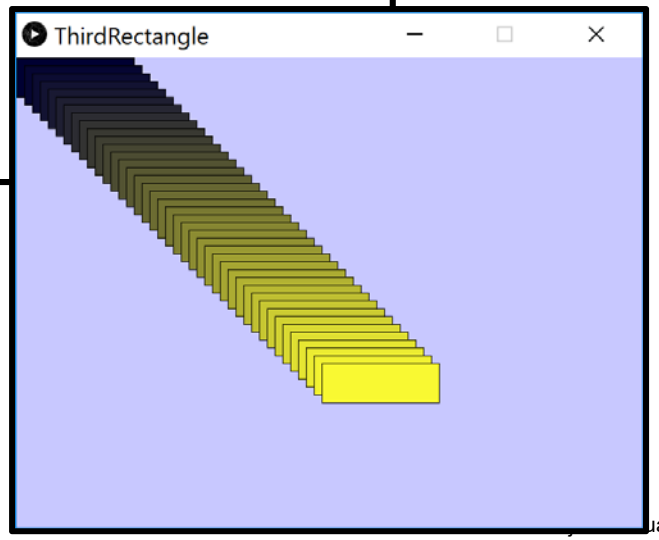
So, for example, if we wanted to turn an x value into a red color, we might say:

int red = int(map(x, 0, width - 1, 0, 255));

More Sophisticated Relationships:

The *map( )* function

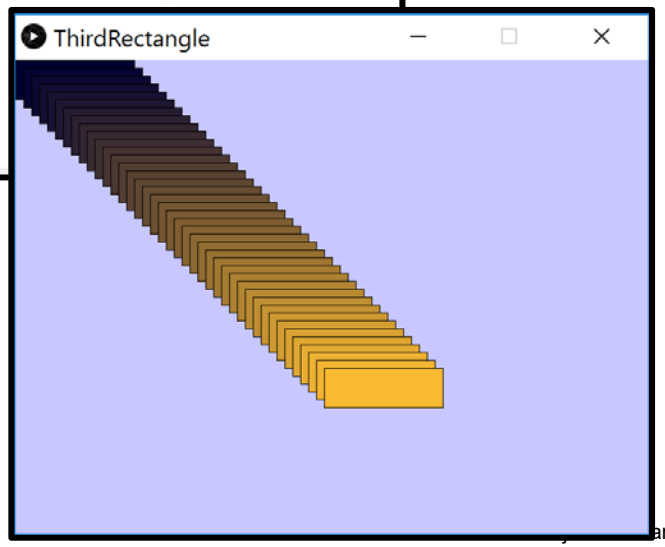
```
void
draw(  )
{
  stroke( 0, 0, 0 );
  fill( 255, 50, 50  );
  for( int x = 0 ; x < 400 ; x = x + 10 )
  {
    int y = x;
    int red   = int( map( x,  0, 399,  0, 255 ) );
    int green = int( map( y,  0, 399,  0, 255 ) );
    //println( "x = " + x + " red = " + red );
    fill( red, green, 50 );
    rect( x, y,  150, 50 );
  }
```



More Sophisticated Relationships:

The *map( )* function

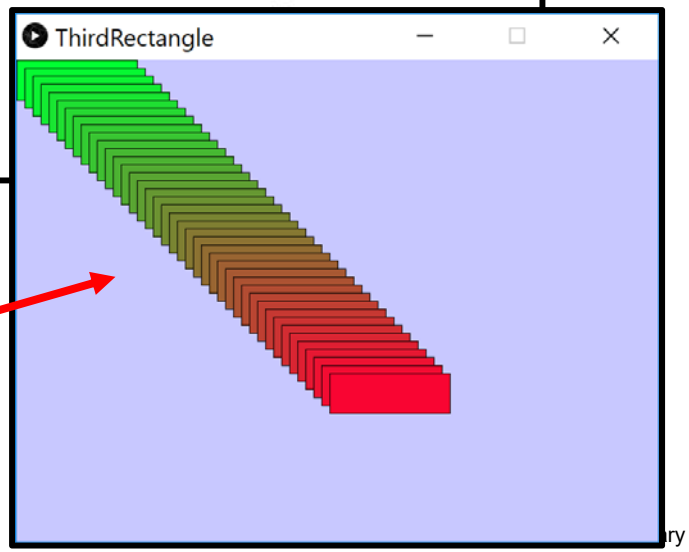
```
void
draw(  )
{
  stroke( 0, 0, 0 );
  fill( 255, 50, 50  );
  for( int x = 0 ; x < 400 ; x = x + 10 )
  {
    int y = x;
    int red   = int( map( x,  0, 399,  0, 255 ) );
    int green = int( map( y,  0, 399,  0, 255 ) );
    green = 3 * green / 4;
    //println( "x = " + x + " red = " + red );
    fill( red, green, 50 );
    rect( x, y,  150, 50 );
  }
```



The *map( )* function can also do blending


```
void
draw(  )
{
  stroke( 0, 0, 0 );
  fill( 255, 50, 50  );
  for( int x = 0 ; x < 400 ; x = x + 10 )
  {
    int y = x;
    int red   = int( map( x,  0, 399,   0, 255 ) );
    int green = int( map( y,  0, 399, 255,   0 ) );
    //println( "x = " + x + " red = " + red );
    fill( red, green, 50 );
    rect( x, y,  150, 50 );
  }
```

Interpolate one forward and the other one backwards



All-green morphs into all-red.