Variables and For-loops

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.

```c
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.

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

Arithmetic operations in programming are:

+ Addition
- Subtraction
* Multiplication
/ Division
() Grouping

Drawing One Rectangle is Pretty Straightforward

```c
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 += 10 ) {
    int y = x;
    rect( x, y, 150, 50 );
  }
}
```

For-loops to the Rescue!

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

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 ) );
```

0 x width - 1

0 red 255
The `map()` function can also do blending.

Interpolate one forward and the other one backwards.

All-green morphs into all-red.