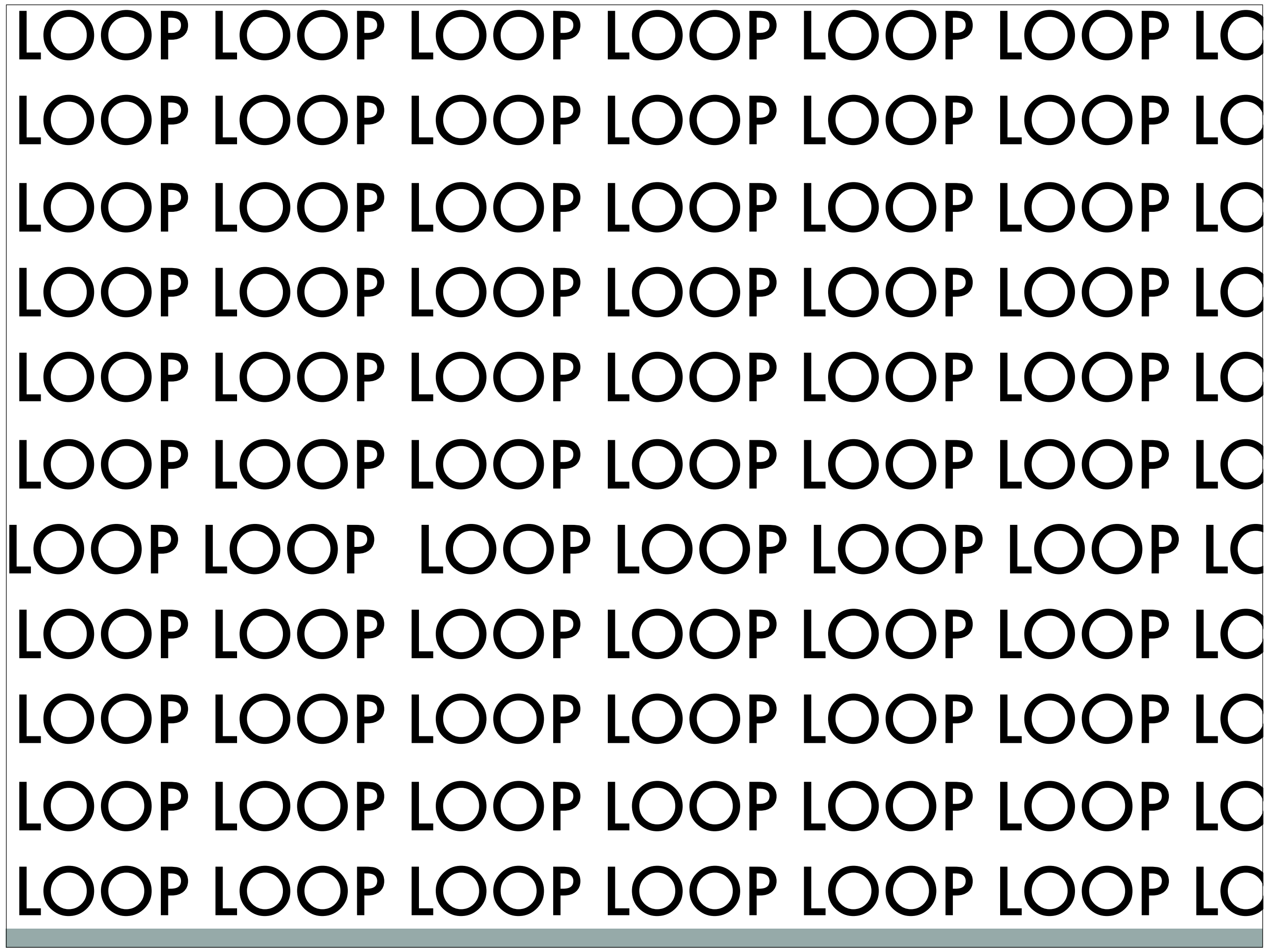
23.10.18 //

PROCESSING IV

DART 631
RILLA KHALED



LOOP LOOP LOOP LOOP LOOP LOOP LO
LOOP LOOP LOOP LOOP LOOP LOOP LO
LOOP LOOP LOOP LOOP LOOP LOOP LO
LOOP LOOP LOOP LOOP LOOP LOOP LO
LOOP LOOP LOOP LOOP LOOP LOOP LO
LOOP LOOP LOOP LOOP LOOP LOOP LO
LOOP LOOP LOOP LOOP LOOP LOOP LO
LOOP LOOP LOOP LOOP LOOP LOOP LO
LOOP LOOP LOOP LOOP LOOP LOOP LO
LOOP LOOP LOOP LOOP LOOP LOOP LO
LOOP LOOP LOOP LOOP LOOP LOOP LO

Now we turn to the idea of controlling the flow of a program

Which is getting pretty sophisticated, and pretty awesome.

OUR FRIEND, REPETITION.

We have already seen how extremely useful repetition is in the form of the `draw()` loop that we use to make time exist in our Processing programs.

But it would be fun to be able to repeat other things as well, right?

HOW TO PAINT THE FENCE.

```
int picketWidth = 20;
int picketHeight = 100;
size(1000,500);
background(0,200,0);

rect(0,400,picketWidth,picketHeight);
rect(20,400,picketWidth,picketHeight);
rect(40,400,picketWidth,picketHeight);
rect(60,400,picketWidth,picketHeight);
rect(80,400,picketWidth,picketHeight);
rect(100,400,picketWidth,picketHeight);
```

And then you die of boredom and you haven’t even painted all that much of the fence anyway.

What a drag.

```
int picketWidth = 20;
int picketHeight = 100;
size(1000,500);
background(0,200,0);

rect(0,400,picketWidth,picketHeight);
rect(20,400,picketWidth,picketHeight);
rect(40,400,picketWidth,picketHeight);
rect(60,400,picketWidth,picketHeight);
rect(80,400,picketWidth,picketHeight);
rect(100,400,picketWidth,picketHeight);
```

`while`ing AWAY THE TIME

```
while ( condition )
{
      // Do something, like painting a picket!
}
```

This will execute the code inside the curly brackets for as long as condition is true.

It will check the condition, find it's true, do something, check the condition, find it's true, do something, check the condition, find it's true, do something…

Until the condition is false.

It's quite a lot like an if statement that repeats the code until the condition is false,

rather than just doing it once when it's true

IF WE WERE PAINTING THE FENCE…

```
while (fenceIsNotPainted)
{
    // Paint one more picket
}
```

That’s the theory anyway, but we’ll need some extra information to make it work out.

IN THE LOOP.

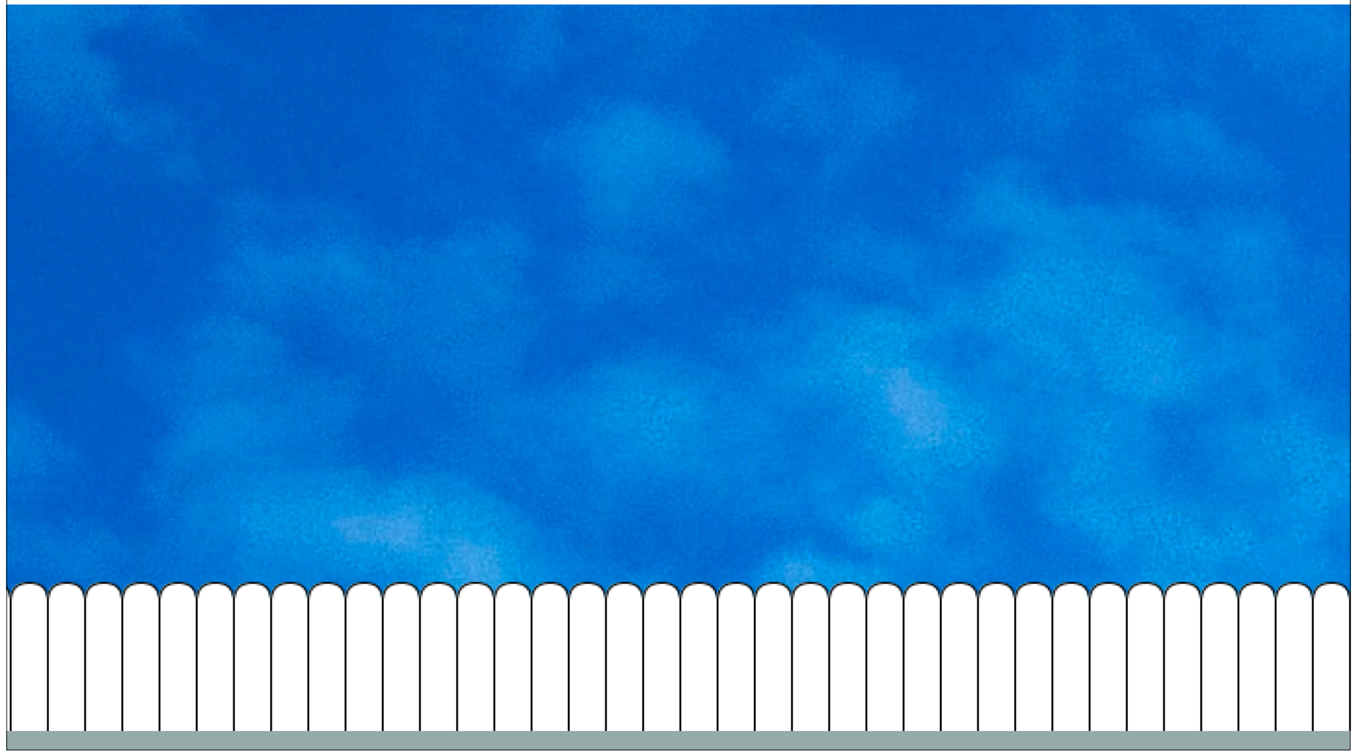
There are three main things you need to establish when you're using a loop:

Start condition. What is the situation you start off in? (In this case, it's that the fence is not painted yet.)

Stop condition. What is the situation when you can stop? (In this case, it's that the fence is painted.)

Action. What should you do each time through the loop? (In this case, it's to paint one picked.)

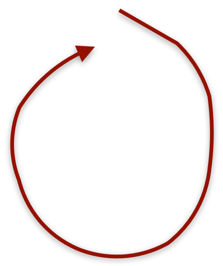
PAINTING THE FENCE IN STYLE...



```
int picketX = 0;
int picketWidth = 20;
int picketHeight = 100;
int picketRounding = 10;
PImage sky;
int skyLoc = -100;
int skyShift = 1;

void setup()
{
  size(1000, 400);
  background(200, 200, 255);
  sky = loadImage("sky.jpg");
}

void draw()
{
  if (frameCount % 20 == 0) {
    skyLoc += skyShift;
    if (skyLoc == 0 || skyLoc == -100) {
      skyShift *= -1;
    }
  }

  image(sky, skyLoc, 0);
  picketX = 0;

  while (picketX < width)
  {
    rect(picketX, 320, picketWidth, picketHeight, picketRounding);
    picketX += picketWidth;
  }
}
```

SEE code/L8/FenceWhile

FENCE-PAINTING WHILE LOOP



```
while (picketX < width)
  {
    rect(picketX, 320, picketWidth, picketHeight, picketRounding);
    picketX += picketWidth;
  }
}
```

Notice that we have to *change* `picketX` or the loop might never end! This is a very important issue with loops – making sure that they will eventually finish and that the condition you're checking becomes false.

The Song That Doesn’t End
http://www.youtube.com/watch?v=HNTxr2NJHa0

The ultimate infinite loop!

TO INFINITY AND BEYOND!

```
while ( true )
{
  singTheSongThatDoesntEnd();
}
```

This is an *infinite loop* – it will never finish. It will freeze up your program.

Not all infinite loops are as obvious as this one!

SOMETIMES OUR LOOP IS ABOUT DOING AN ACTION SOME NUMBER OF TIMES

```
x = 0
while ( x < y )
{
  do action
  x = x+1
}
```

`for` WHAT IT'S WORTH…

```
for (int i = 0; i < 10; i++)
{
      println("Now i is " + i);
}
```

This is a `for` loop that counts from 0 up to 9 and prints out each number in the form “Now i is 0”, “Now i is 1”, Now i is 2” and so on…

Let’s look at this weird syntax.

```
for (int i = 0; i < 10; i++)
{
  println("Now i is " + i);
}
```

for WHAT IT’S WORTH…

```
for (int i = 0; i < 10; i++)

{

    println(“Now i is “ + i);

}
```

Here is the word that tells Processing we’re going to do a `for` loop.

for WHAT IT’S WORTH...

```
for (int i = 0; i < 10; i++)

{

     println(“Now i is “ + i);

}
```

This is our starting condition. It’s very common to declare a counting variable here, and it’s very common to call it ‘`i`’.

So this bit will happen *once* right at the start of the loop (a lot like our `setup()` function in Processing).

i stands for iterator, in the sense of “iterate through this set of things”

`for` WHAT IT'S WORTH...

```
for (int i = 0; i < 10; i++)
{
     println("Now i is " + i);
}
```

This is the condition we will check each time through the loop, just like with a `while` loop.

It does get checked *before* the loop even starts the first time.

As soon as it is false, the loop stops.

for WHAT IT'S WORTH...

```
for (int i = 0; i < 10; i++)
{
    println("Now i is " + i);
}
```

This is the usual use of curly brackets to mark out the code you want to run each time through the `for` loop.

`for` WHAT IT'S WORTH…

```
for (int i = 0; i < 10; i++)
{
      println("Now i is " + i);
}
```

This is the update statement that will get run *after* each pass through the loop – so we are adding one to `i` after each pass through.

It's very common for this to simply add one to the "iterator" (`i`), but you can do anything here if you want.

`for` WHAT IT'S WORTH...

This happens only once at the beginning.

1. This happens first for each loop.

3. This happens after each loop.

```
for (int i = 0; i < 10; i++)
{
    println("Now i is " + i);
}
```

2. This happens each time the condition comes out true.

OTHER `for`s

```
for (int i = 0; i < 10; i++)
```

```
for (int i = 20; i > 0; i--)
```

```
for (int i = 0; i <= 100; i+=10)
```

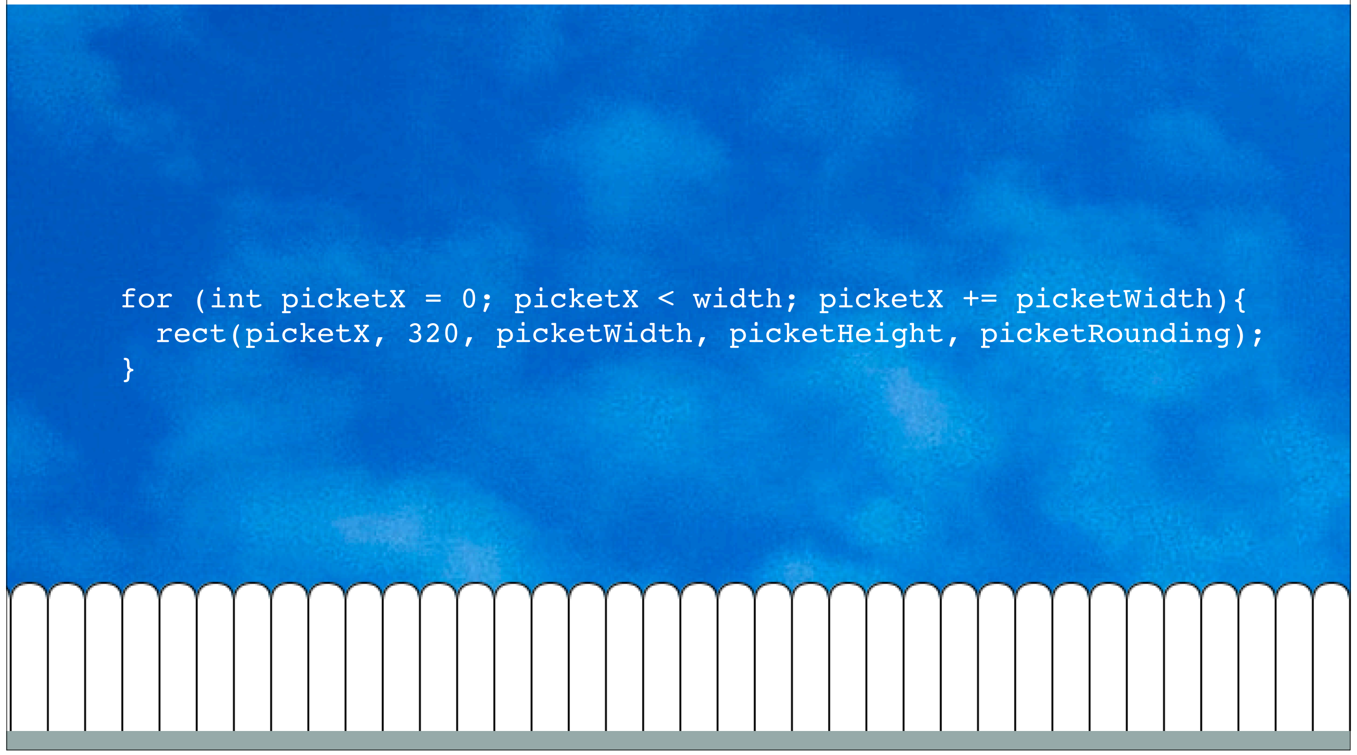
And so on. You can do pretty much whatever you want so long as you make sure you have a start condition, an end condition, and some useful update statement.

`for` AND `while` ARE THE SAME

```
int i = 0;
while (i < 10)
{
        println(i);
        i++;
}

for (int i = 0; i < 10; i++)
{
        println(i);
}
```

These are exactly the same thing.

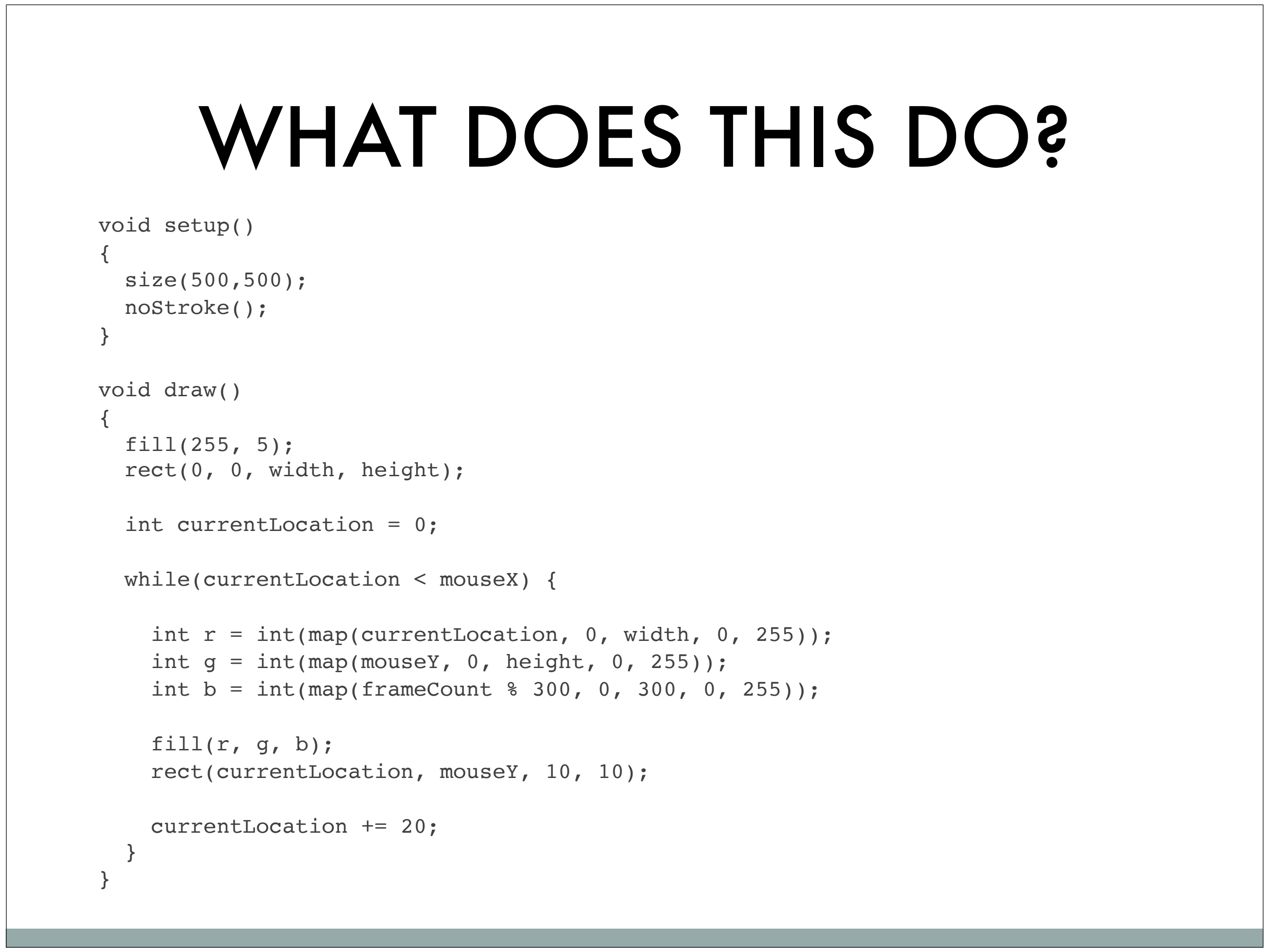
PAINTING THE FENCE REAL SMALL!



```
for (int picketX = 0; picketX < width; picketX += picketWidth){
  rect(picketX, 320, picketWidth, picketHeight, picketRounding);
}
```

```
int picketX = 0;
int picketWidth = 20;
int picketHeight = 100;
int picketRounding = 10;
PImage sky;
int skyLoc = -100;
int skyShift = 1;

void setup()
{
  size(1000, 400);
  background(200, 200, 255);
  sky = loadImage("sky.jpg");
}

void draw()
{
  if (frameCount % 20 == 0) {
    skyLoc += skyShift;
    if (skyLoc == 0 || skyLoc == -100) {
      skyShift *= -1;
    }
  }

  image(sky, skyLoc, 0);

  for (int picketX = 0; picketX < width; picketX += picketWidth){
    rect(picketX, 320, picketWidth, picketHeight, picketRounding);
  }
}
```

SEE code/L8/FenceFor

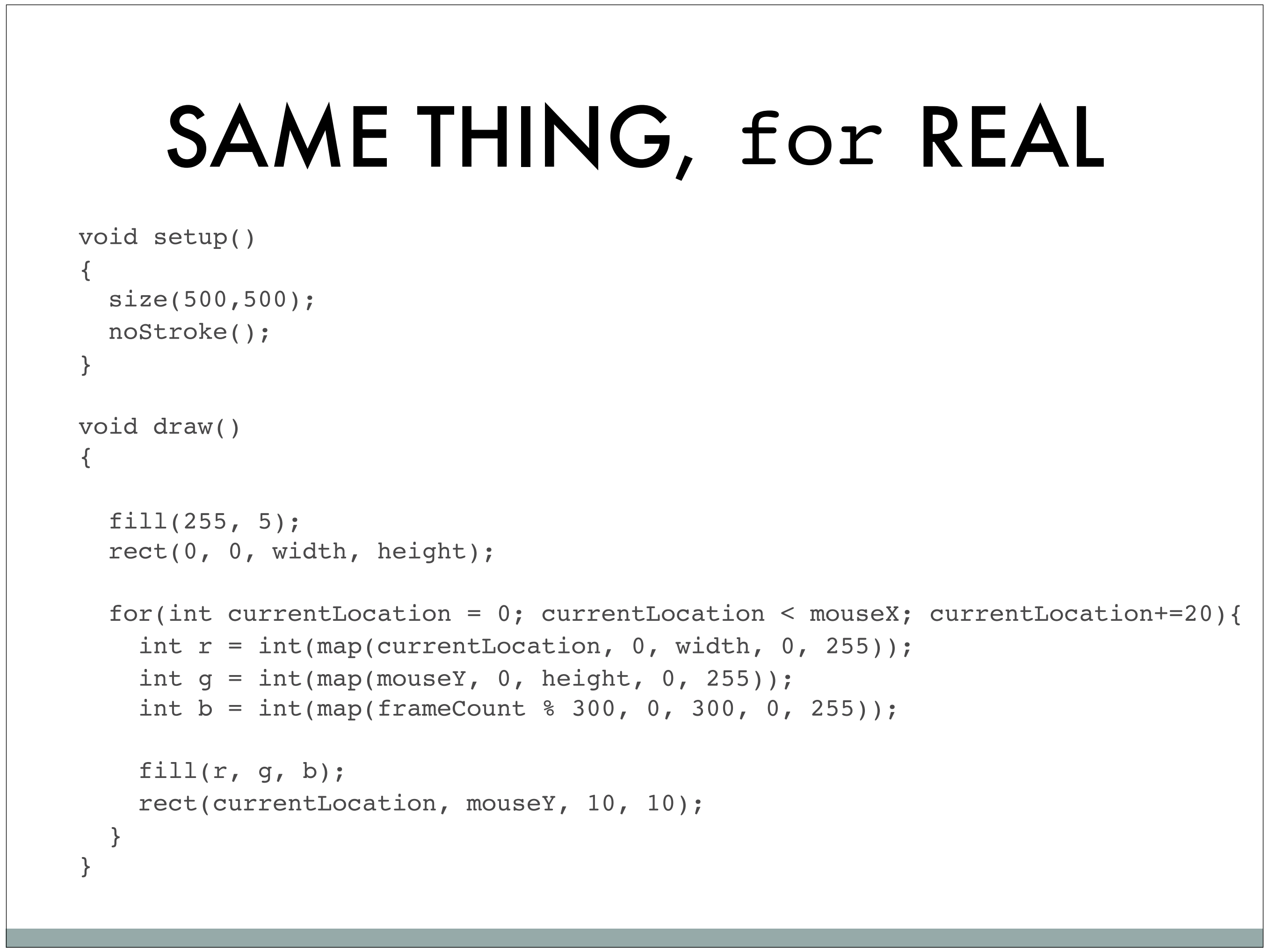


WHAT DOES THIS DO?

```
void setup()
{
  size(500,500);
  noStroke();
}

void draw()
{
  fill(255, 5);
  rect(0, 0, width, height);

  int currentLocation = 0;

  while(currentLocation < mouseX) {

    int r = int(map(currentLocation, 0, width, 0, 255));
    int g = int(map(mouseY, 0, height, 0, 255));
    int b = int(map(frameCount % 300, 0, 300, 0, 255));

    fill(r, g, b);
    rect(currentLocation, mouseY, 10, 10);

    currentLocation += 20;
  }
}
```

```
void setup()
{
  size(500,500);
  noStroke();
}

void draw()
{
  /* draw a white rectangle with quite a lot of transparency
  to cover the screen */
  fill(255, 5);
  rect(0, 0, width, height);

  // declare a variable called currentLocation corresponding to
  // the X loc of a rect
  int currentLocation = 0;

  // while we have not hit mouseX
  while(currentLocation < mouseX) {

    // establish a mapped color

    // R: based on currentLocation
    int r = int(map(currentLocation, 0, width, 0, 255));

    // G: based on mouseY
    int g = int(map(mouseY, 0, height, 0, 255));

    // B: based on how many frames have elapsed
    int b = int(map(frameCount % 300, 0, 300, 0, 255));

    fill(r, g, b);

    // draw a rectangle at currentLocation, mouseY that is 10x10
    rect(currentLocation, mouseY, 10, 10);

    // update currentLocation - our next rect drawing X loc - by 20
    currentLocation += 20;
  }
}
```

See code/L8/ImitationRichterWhile



SAME THING, `for` REAL

```
void setup()
{
  size(500,500);
  noStroke();
}

void draw()
{

  fill(255, 5);
  rect(0, 0, width, height);

  for(int currentLocation = 0; currentLocation < mouseX; currentLocation+=20){
    int r = int(map(currentLocation, 0, width, 0, 255));
    int g = int(map(mouseY, 0, height, 0, 255));
    int b = int(map(frameCount % 300, 0, 300, 0, 255));

    fill(r, g, b);
    rect(currentLocation, mouseY, 10, 10);
  }
}
```

```
void setup()
{
  size(500,500);
  noStroke();
}

void draw()
{
  /* draw a white rectangle with quite a lot of transparency
  to cover the screen */
  fill(255, 5);
  rect(0, 0, width, height);

  /* until we run into the mouseX location, do the following
  and keep updating the val of currentLocation */
  for(int currentLocation = 0; currentLocation < mouseX; currentLocation+=20){

    // establish a mapped color

    // R: based on currentLocation
    int r = int(map(currentLocation, 0, width, 0, 255));

    // G: based on mouseY
    int g = int(map(mouseY, 0, height, 0, 255));

    // B: based on how many frames have elapsed
    int b = int(map(frameCount % 300, 0, 300, 0, 255));

    fill(r, g, b);

    // draw a rectangle at currentLocation, mouseY that is 10x10
    rect(currentLocation, mouseY, 10, 10);
  }
}
```

See code/L8/ImitationRichterFor

BASIC PROCESSING EXERCISE

Using everything that we have learned in Processing so far, make an interactive scene that can represent day and night.

The scene should use

- random()
- mouseX and mouseY
- mouse input or keyboard input
- at least 3 images
- Conditional statements: if, else if, else (etc.)
- a for loop or a while loop

I will also be paying attention to how tidy your program is, whether you have commented your code, etc.

This is due on **30th October at 1:30pm.**