# Loops and Introduction to Functions

CS 110

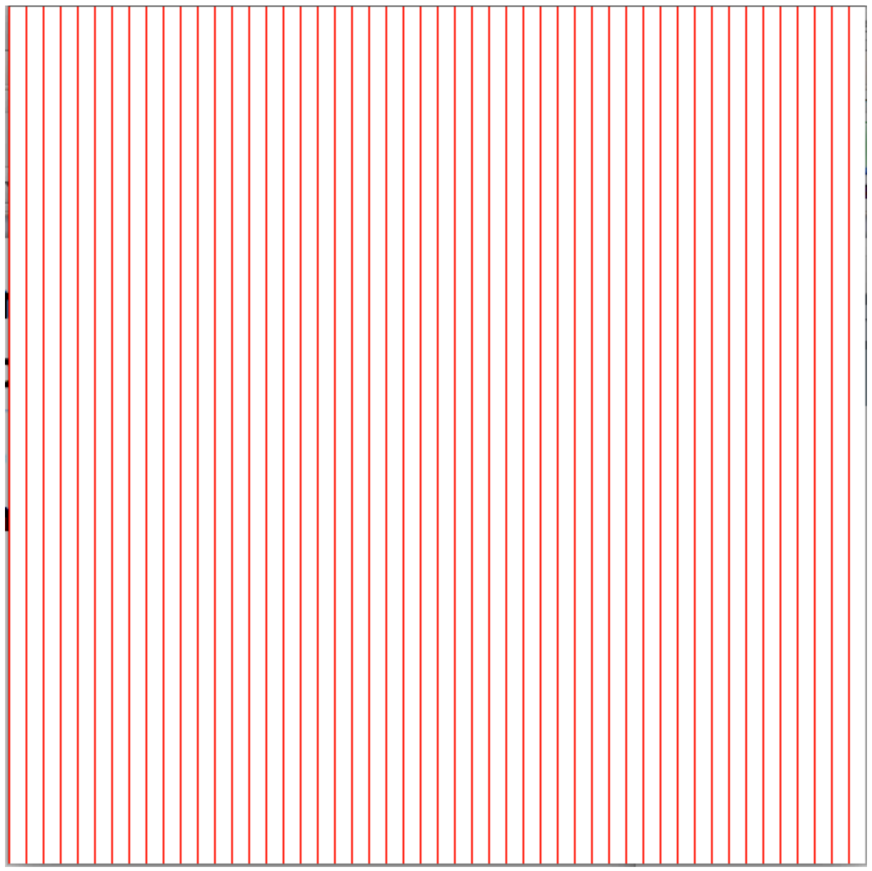
Eric Eaton and Paul Ruvolo

## Quick Review

- Switch statement
- Loops

## Doing something different in each "iteration" of the loop

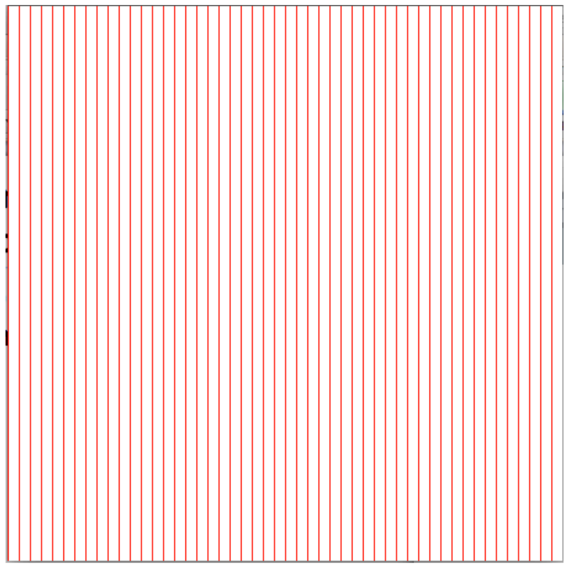
How would I write code to generate the following image in processing?



## Solution

```
size(500,500);
background(255);
int x = 0;

while (x < width) {
    stroke(255,0,0);
    line(x,0,x,height);
    x = x + 5;
}
```



## Example of a Loop

```
…
int myNumber = 6;
int factorial = 1;
while ( myNumber > 0 ) {
    factorial *= myNumber;
    --myNumber;
}
println(factorial);
```

## The 3 Parts of a Loop

…  
int i = 1 ; ← initialization of **loop control variable**

// count from 1 to 100  
while ( i < 101 ) { ← test of **loop termination condition**  
  println( i ) ;  
  i = i + 1 ; ← modification of loop control variable  
}

## The for Loop Repetition Structure

- The **for** loop handles details of the counter-controlled loop “automatically”.
- The initialization of the the loop control variable, the termination condition test, and control variable modification are handled in the **for** loop structure.

```
for (int i = 1; i < 101; i = i + 1) {
               ↑          ↑           ↑
  initialization          |       modification
}                        test
```

## *for* Loop Examples

- A *for* loop that counts from 0 to 9:

```
// modify part can be simply “i++”
for ( i = 0;  i < 10;  i = i + 1 ) {
 System.out.println( i ) ;
}
```

- …or we can count backwards by 2’s :

```
// modify part can be “i -= 2”
for ( i = 10;  i > 0;  i = i - 2 ) {
 System.out.println( i ) ;
}
```

## When Does a *for* Loop Initialize, Test and Modify?

- Just as with a while loop, a for loop
  - initializes the loop control variable before beginning the first loop iteration
  - performs the loop termination test before each iteration of the loop
  - modifies the loop control variable at the very end of each iteration of the loop
- The for loop is easier to write and read for counter-controlled loops.

```
void setup() {
  size(500, 500);
  smooth();

  float diameter = 500.0;
  while ( diameter > 1.0 ) {
   ellipse( 250, 250, diameter, diameter);
   diameter = diameter - 10.0;
  }
}

void draw() { }
```

---

```
void setup() {
  size(500, 500);
  smooth();

  for (float diameter = 500.0; diameter > 1.0; diameter -= 10.0 )
  {
    ellipse( 250, 250, diameter, diameter);
  }
}

void draw() { }
```

## The *break & continue* Statements

- The `break` & `continue` statements can be used in **while** and **for** loops to cause the remaining statements in the body of the loop to be skipped; then:
  - `break` causes the looping itself to abort, while…
  - `continue` causes the next turn of the loop to start. In a **for** loop, the modification step will still be executed.

## Example break in a for Loop

```
…
int i;
for (i = 1; i < 10; i = i + 1) {
     if (i == 5) {
         break;
     }
     System.out.println(i);
}
System.out.println(“\nBroke out of loop at i = “ + i);
```

- **<u>OUTPUT</u>:**
- **1 2 3 4**
- **Broke out of loop at i = 5.**

## Example continue in a for Loop

```
…
int i;
for (i = 1; i < 10; i = i + 1) {
    if (i == 5) {
        continue;
    }
    System.out.println(i);
}
System.out.println(“Done”);
```

## Example continue in a for Loop

```
…
int i;
for (i = 1; i < 10; i = i + 1) {
     if (i == 5) {
         continue;
     }
     System.out.println(i);
}
System.out.println(“Done”);
```

**<u>OUTPUT</u>:**

```
1 2 3 4 6 7 8 9

Done.
```

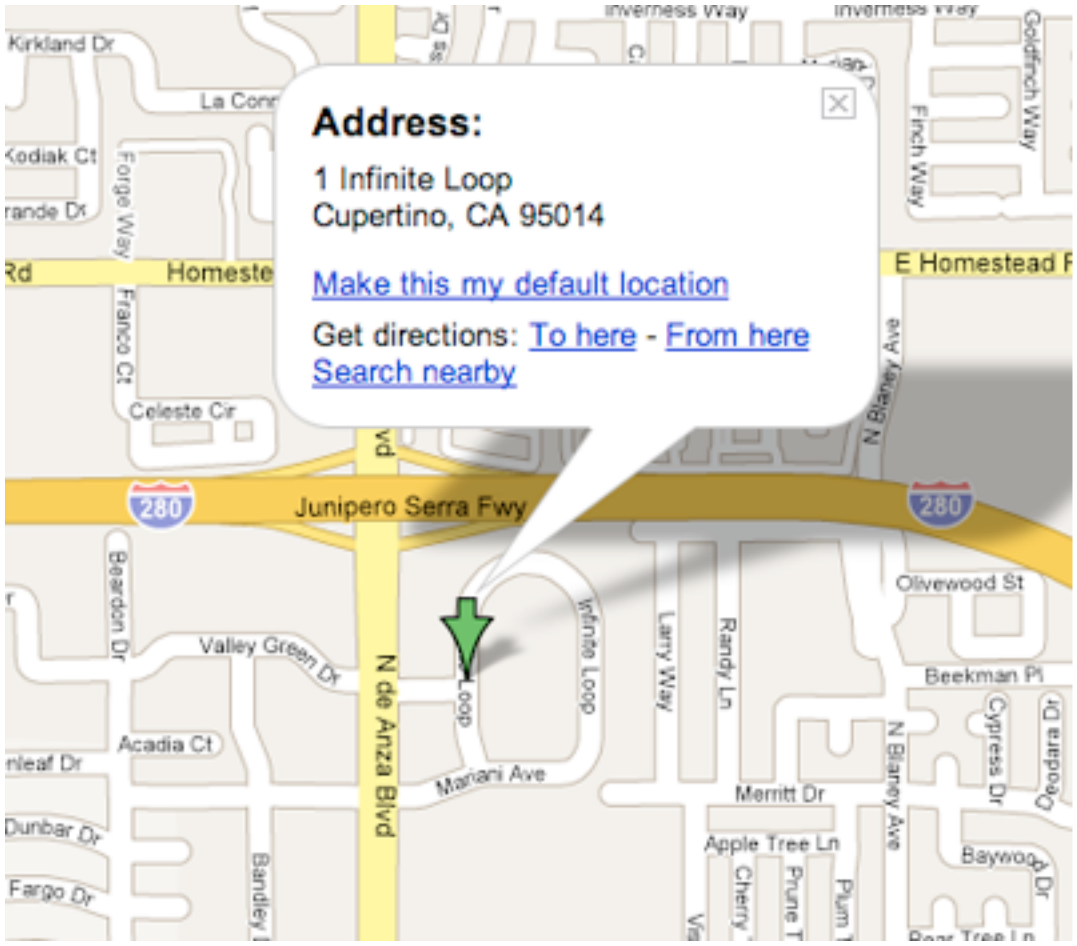
## Problem: continue in while Loop

```
// This seems equivalent to for loop
// in previous slide—but is it??
…
int i = 1;
while (i < 10) {
    if (i == 5) {
        continue;
    }
    System.out.println(i);
    i = i + 1;
}
System.out.println(“Done”);
```

**<u>OUTPUT</u>:**

**???**

## Infinite Loop



**Address:**

1 Infinite Loop
Cupertino, CA 95014

Make this my default location

Get directions: To here - From here
Search nearby

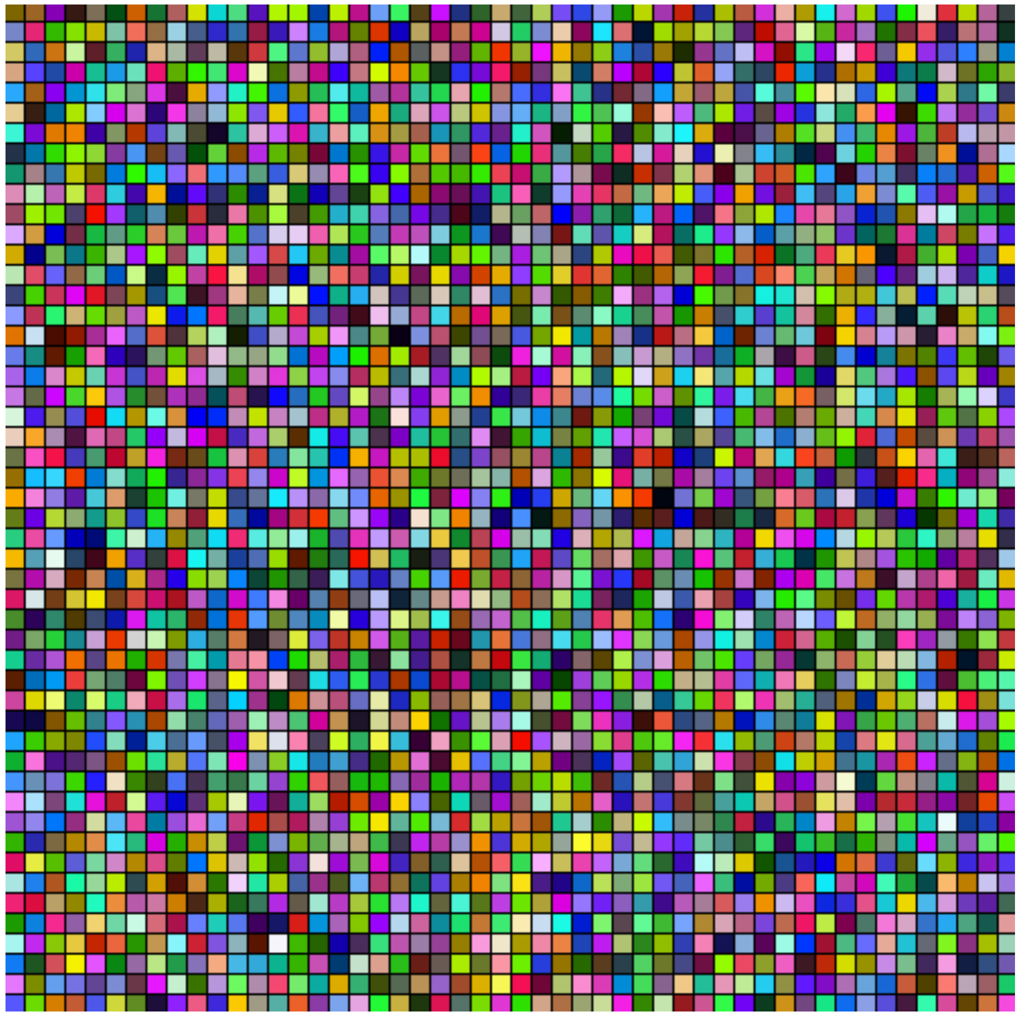
## Nested Loops

Where can we use loops?  Inside any block of code!

This means, that we can use loops inside of other loops!

What type of picture might we want to use nested loops to draw?

# Crazy Checkerboard!



## Crazy Checkerboard Code

```
size(500,500);
int x = 0;
int y = 0;

for (x = 0; x < width; x += 10) {
   for (y = 0; y < height; y += 10) {
      fill(random(0,255),random(0,255),random(0,255));
      rect(x,y,10,10);
   }
}
```

## Another Example of Nested Loops

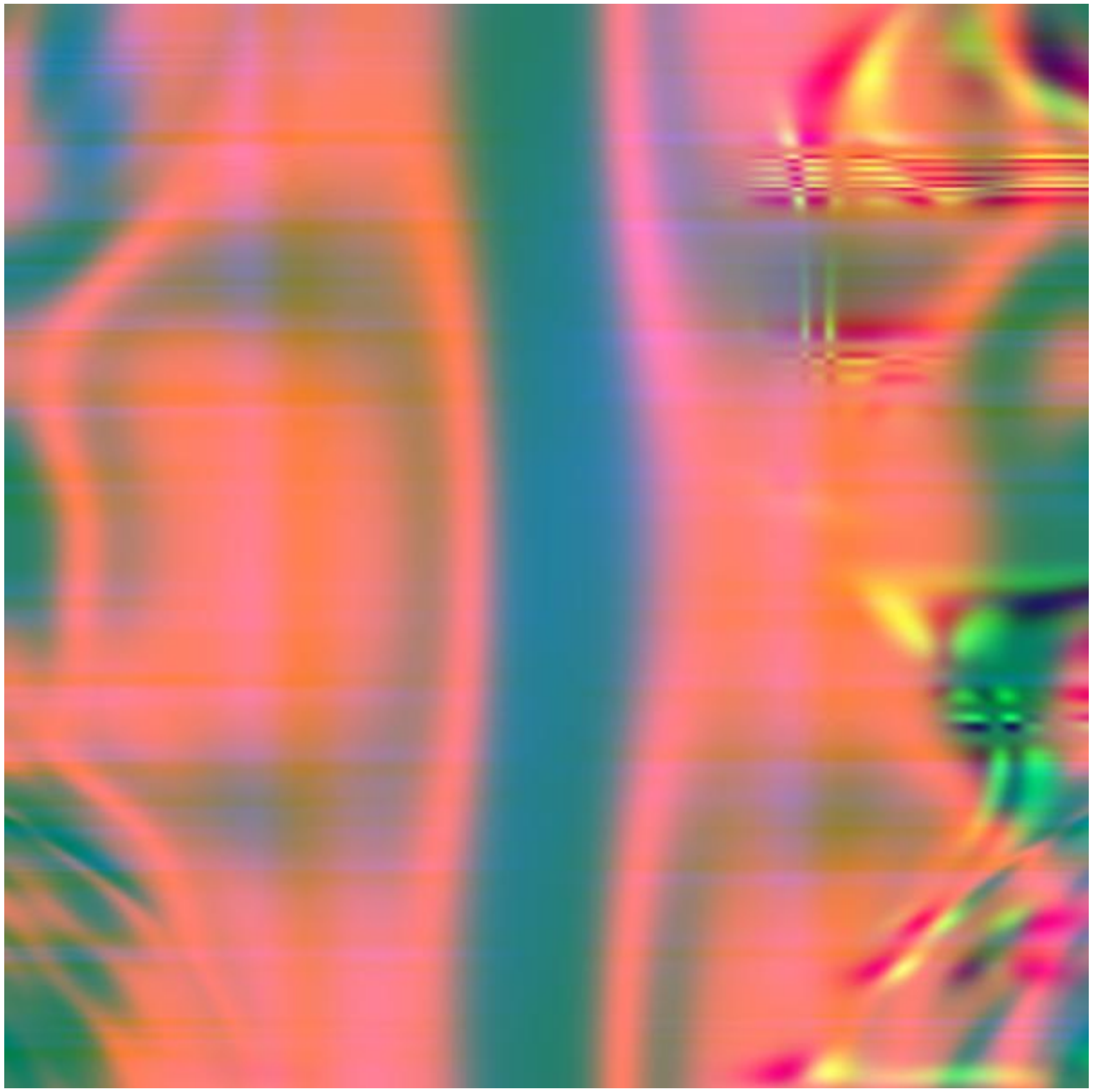
```
float delta = 5.0;
float factor = 0.0;

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

void draw() {
  factor+=0.2;
  noStroke();
  float r;
  float c;
  for (r=0.0; r<height; r+=delta) {
    for (c=0.0; c<width; c+=delta) {

      // Use factor to scale shape
      float x = map(c, 0.0, 500.0, 0.0, 3.0*TWO_PI);
      float y = map(r, 0.0, 500.0, 0.0, 3.0*TWO_PI);
      float shade = map(sin(factor)*sin(x)*sin(y), -1.0, 1.0, 0, 255);
      fill( shade );
      rect(r, c, delta, delta);
    }
  }
}
```

## A More Advanced Version of the Previous Code



## Variable Scope

***Variable scope*:**

- That set of code statements in which the variable is known to the compiler
- Where it can be referenced in your program.
- Limited to the ***code block*** in which it is defined.
  - A ***code block*** is a set of code enclosed in braces (***{ }***).

One interesting application of this principle allowed in Java involves the for loop construct.

## for-loop index

- Can declare and initialize variables in the heading of a for loop.
- These variables are local to the for-loop.
- They may be reused in other loops.

```
String s = “hello world”;
int count = 1;
for (int i = 0; i < s.length(); i++)
{
   count *= 2;
}
//using 'i' here generates a compiler error
```

## Intro to Functions

```
float w, h;
void setup() {
  size(500, 500);
  smooth();
  w = 50;
  h = 50;
}

void draw() {
  strokeWeight(15);
  stroke(113, 71, 71);
  line(392, height, 400, 0);
  line(395, height/2, 200, 0);
}

// function continued on next page
```

## Intro to Functions

```
void keyPressed() {
  h = 25;
  noStroke();
  fill(175, 175, 0);
  beginShape();
  curveVertex(mouseX, mouseY);
  curveVertex(mouseX, mouseY);
  curveVertex(mouseX-0.3*h, mouseY - 0.7*h);
  curveVertex(mouseX-w, mouseY-h);
  curveVertex(mouseX-0.7*h, mouseY - 0.2*h);
  curveVertex(mouseX, mouseY);
  curveVertex(mouseX, mouseY);
  endShape(CLOSE);
}

// function continued on next page
```

## Intro to Functions

```
void mousePressed() {
  h = 50;
  noStroke();
  fill(175, 0, 0);
  beginShape();
  curveVertex(mouseX, mouseY);
  curveVertex(mouseX, mouseY);
  curveVertex(mouseX-0.3*h, mouseY - 0.7*h);
  curveVertex(mouseX-w, mouseY-h);
  curveVertex(mouseX-0.7*h, mouseY - 0.2*h);
  curveVertex(mouseX, mouseY);
  curveVertex(mouseX, mouseY);
  endShape(CLOSE);
}
```

## Why is coding in this way a bad idea?

## Functions to the Rescue

Modularity

- Functions allow the programmer to break down larger programs into smaller parts.
- Promotes organization and manageability.

Reuse

- Enables the reuse of code blocks from arbitrary locations in a program.

## Functions

- A function names a block of code, making it reusable.
- Arguments can be "passed in" to function and used in body.
- Arguments are a comma-delimited set of variable declarations.
- Argument values are ***copies*** of passed values, not originals.
- Function must return a value that matches function declaration.

```
return_type function_name( argument_decl_list ) {
    statements;
    return value;
}
```

## What happens when we call a function?

1. The argument expressions are evaluated.
2. The resulting values are copied into the corresponding parameters.
3. The statements in the function's body are evaluated in order.
4. When a return statement is evaluated, the value of its argument is used as the function's result.
5. The calling function continues after "substituting" the function's returned value in place of the function call.

## Built-in Functions vs. User-defined

- Similar to variables where we have user-defined and built-in variables, we have the same in Processing
- Examples of functions built-in to Processing:
  - line() ellipse() curve(), etc.
- Just as we defined our own variables we can define our own functions, however, it is our job to give the functions meaning!

## Our First Function

```
void myRectangle(float x, float y, float rectHeight, float rectWidth) {
    stroke(0);
    line(x,y,x,y+rectHeight);
    line(x,y+rectHeight,x+rectWidth,y+rectHeight);
    line(x+rectWidth,y+rectHeight,x+rectWidth,y);
    line(x,y,x+rectWidth,y);
}
```

## Calling Our Function

```
void myRectangle(float x, float y, float rectHeight, float rectWidth)
{
    stroke(0);
    line(x,y,x,y+rectHeight);
    line(x,y+rectHeight,x+rectWidth,y+rectHeight);
    line(x+rectWidth,y+rectHeight,x+rectWidth,y);
    line(x,y,x+rectWidth,y);
}

void setup() {
   size(500,500);
   background(255);
   myRectangle(100,100,150,50);
}
```