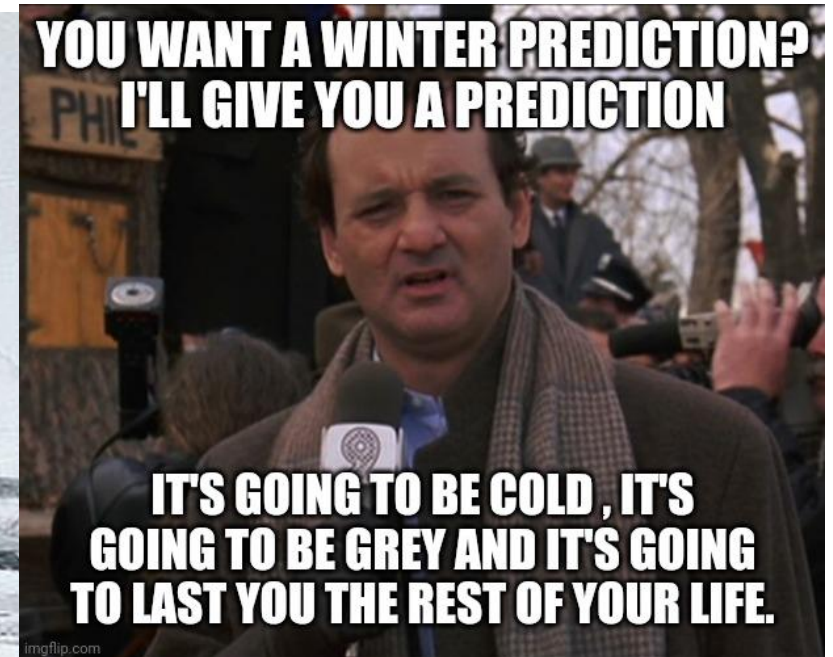


REPETITION

UNIVERSITY OF MOUNT UNION





REPETITION, OR *ITERATION*, IS THE MOST IMPORTANT CONCEPT WE'LL STUDY IN THIS CLASS

- The key to the power of computer programming is to recognize repetitive patterns and *generalize* those patterns
- Example: print the numbers 1-10 in the Console
- The only way we could do this knowing what we know now would be to use 10 separate `println()` statements

THE BRUTE FORCE SOLUTION

```
SKETCH_220130a
1 println(1);
2 println(2);
3 println(3);
4 println(4);
5 println(5);
6 println(6);
7 println(7);
8 println(8);
9 println(9);
10 println(10);
11
```

THE BRUTE FORCE SOLUTION: LIMITATIONS

- The brute force solution is correct...but what if we wanted all the numbers from 1 to 1000?
- Brute force involves way too much work (and way too much code!)
- Instead, we should look for a repetitive pattern
- Here, we want to do *the same thing* to each of the values starting with 1 and going up to 10 (print them to the console)

REPETITION IN PROGRAMMING: THE KEY

- We need to write the operation to be performed in a *general* way (using a **variable**)
- Here, the operation to be performed is
`println(x);`
- We want to do that for all the values from 1 to 10

USING A VARIABLE IN THIS SOLUTION

```
1 int x = 1;  
2  
3 println(x);  
4 x += 1;  
5 println(x);  
6 x += 1;  
7 println(x);  
8 x += 1;  
9 println(x);  
10 x += 1;  
11 println(x);  
12 x += 1;  
13 println(x);  
14 x += 1;  
15 println(x);  
16 x += 1;  
17 println(x);  
18 x += 1;  
19 println(x);  
20 x += 1;  
21 println(x);  
22  
23
```

WHAT OPERATIONS ARE BEING REPEATED?

- Done **once** at the beginning:

```
int x = 1;
```

- Done **ten times** in a row:

```
println(x);
```

```
x += 1;
```

THE `for` STATEMENT IN PROCESSING

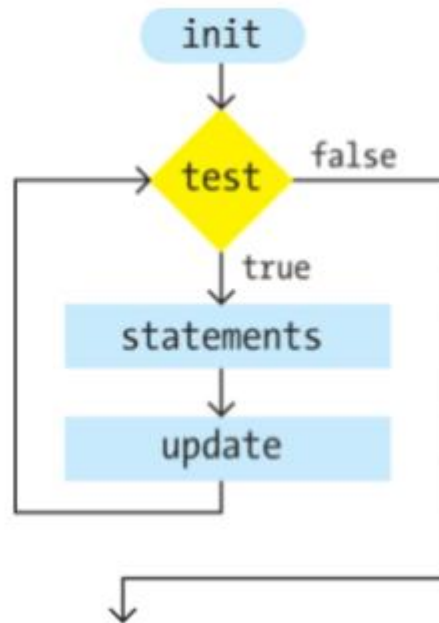
- Purpose: repeatedly execute a block of code a specific number of times

```
for (int x = 1; x <= 10; x++) {  
    println(x);  
}
```

- What if we wanted all the numbers from 1 to 1000? Much easier this way!

** `x++` could also be written as `x+=1` or `x = x + 1` ... they all work!

THE `for` STATEMENT IN PROCESSING: THE LOGIC



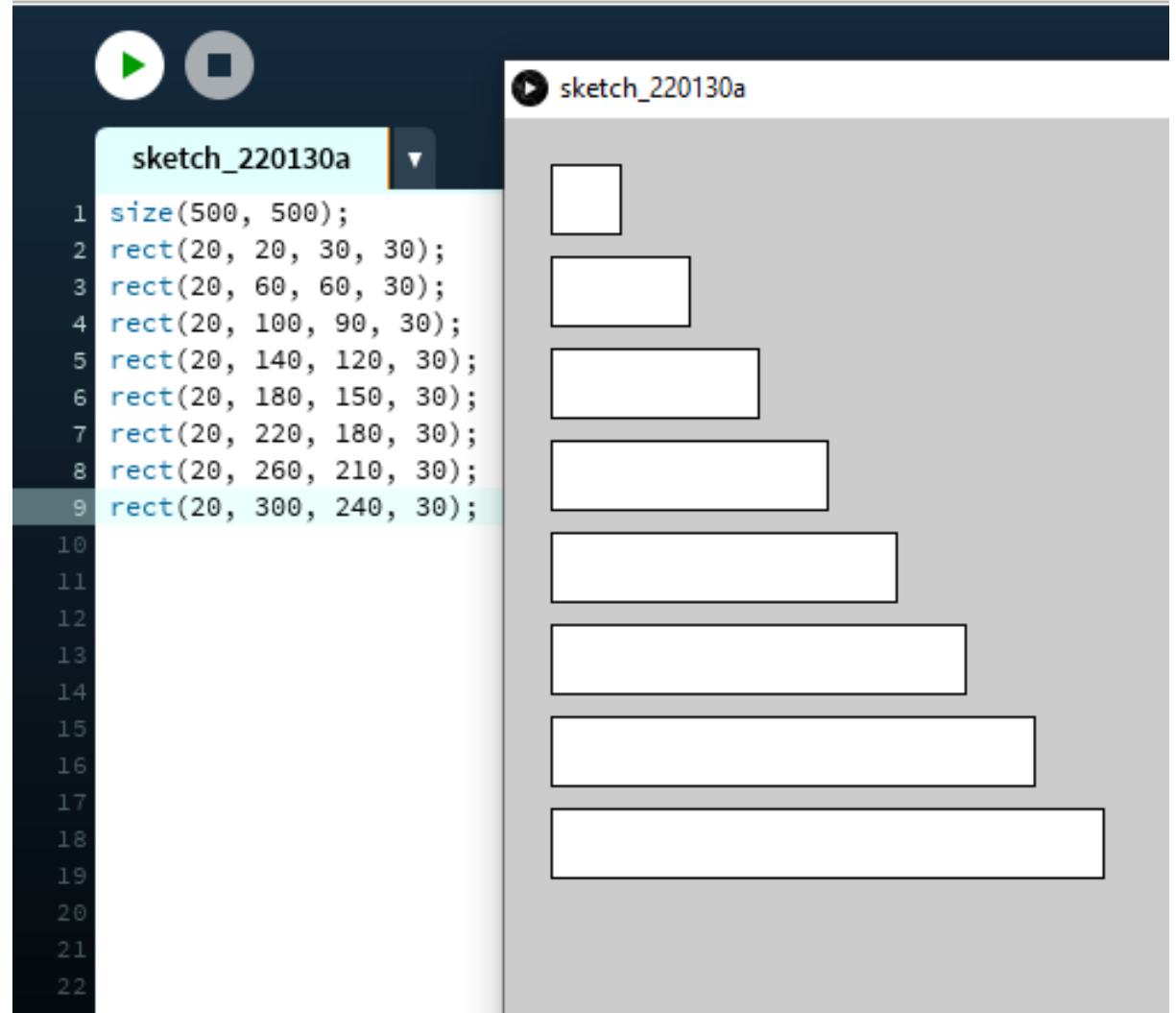
```
for (init; test; update) {  
    statements  
}
```

WHERE'S THE REPETITION?



WHERE'S THE REPETITION IN THE CODE?

- Consider this example – what does it do?
- What is changing from line to line, and what is staying the same?



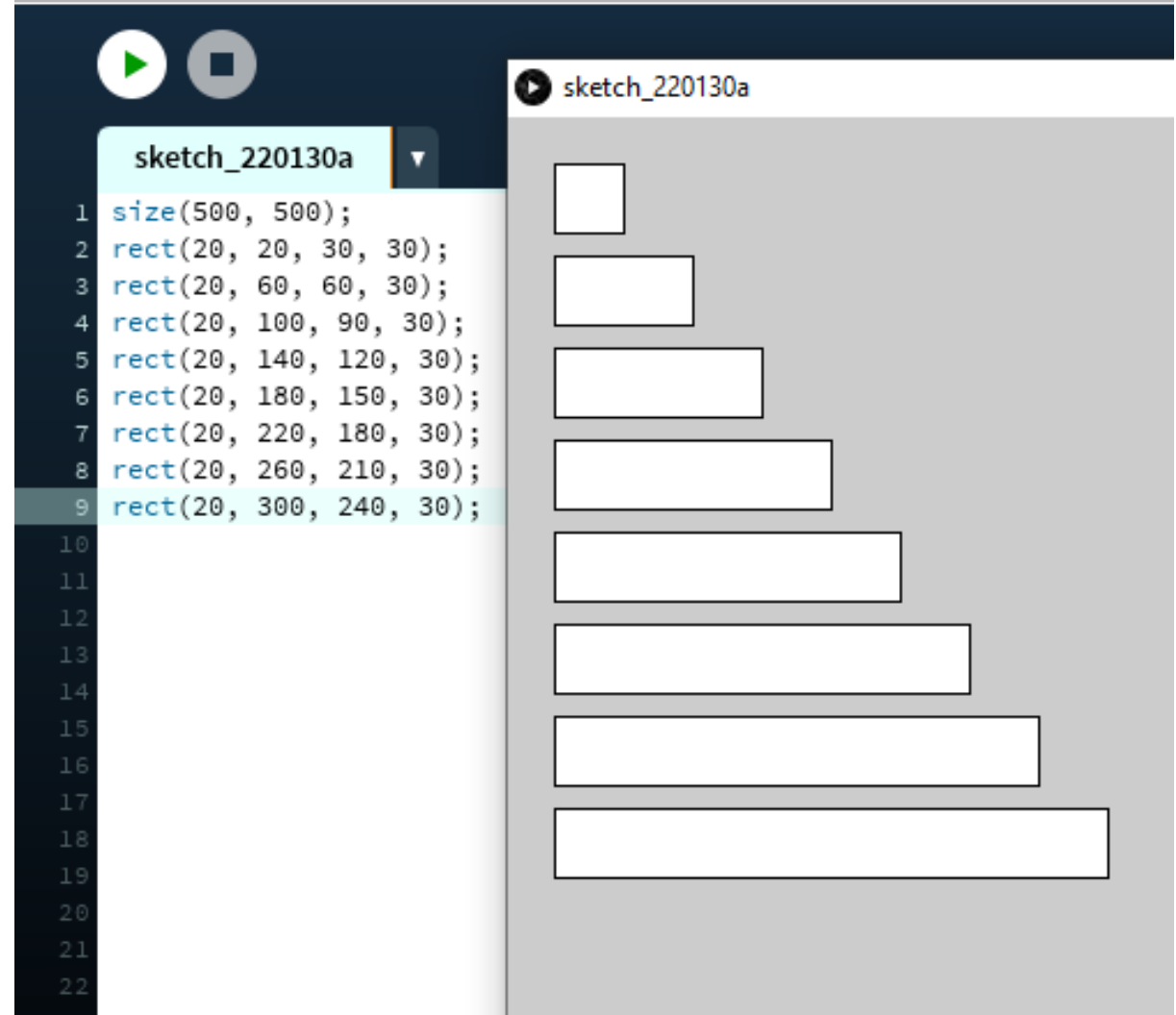
The screenshot shows a code editor window titled "sketch_220130a" with a play button and a stop button at the top. The code is as follows:

```
1 size(500, 500);  
2 rect(20, 20, 30, 30);  
3 rect(20, 60, 60, 30);  
4 rect(20, 100, 90, 30);  
5 rect(20, 140, 120, 30);  
6 rect(20, 180, 150, 30);  
7 rect(20, 220, 180, 30);  
8 rect(20, 260, 210, 30);  
9 rect(20, 300, 240, 30);  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22
```

To the right of the code editor is a preview window titled "sketch_220130a" showing the output of the code. It displays a series of white rectangles on a gray background. The rectangles are all 30 pixels high and 30 pixels wide, starting at x=20. The y-coordinates of the top-left corners are 20, 60, 100, 140, 180, 220, 260, and 300. The x-coordinates of the top-right corners are 50, 90, 130, 170, 210, 250, 290, and 330. This visualizes the pattern in the code where the x-coordinate increases by 30 and the y-coordinate increases by 40 in each iteration.

WHAT'S CHANGING? WHAT'S THE SAME?

- The **vertical position** of each rectangle changes
- The **width** of each rectangle changes
- But the **horizontal position** and **length** of each rectangle stays the same
- How can we generalize this idea and use repetition?



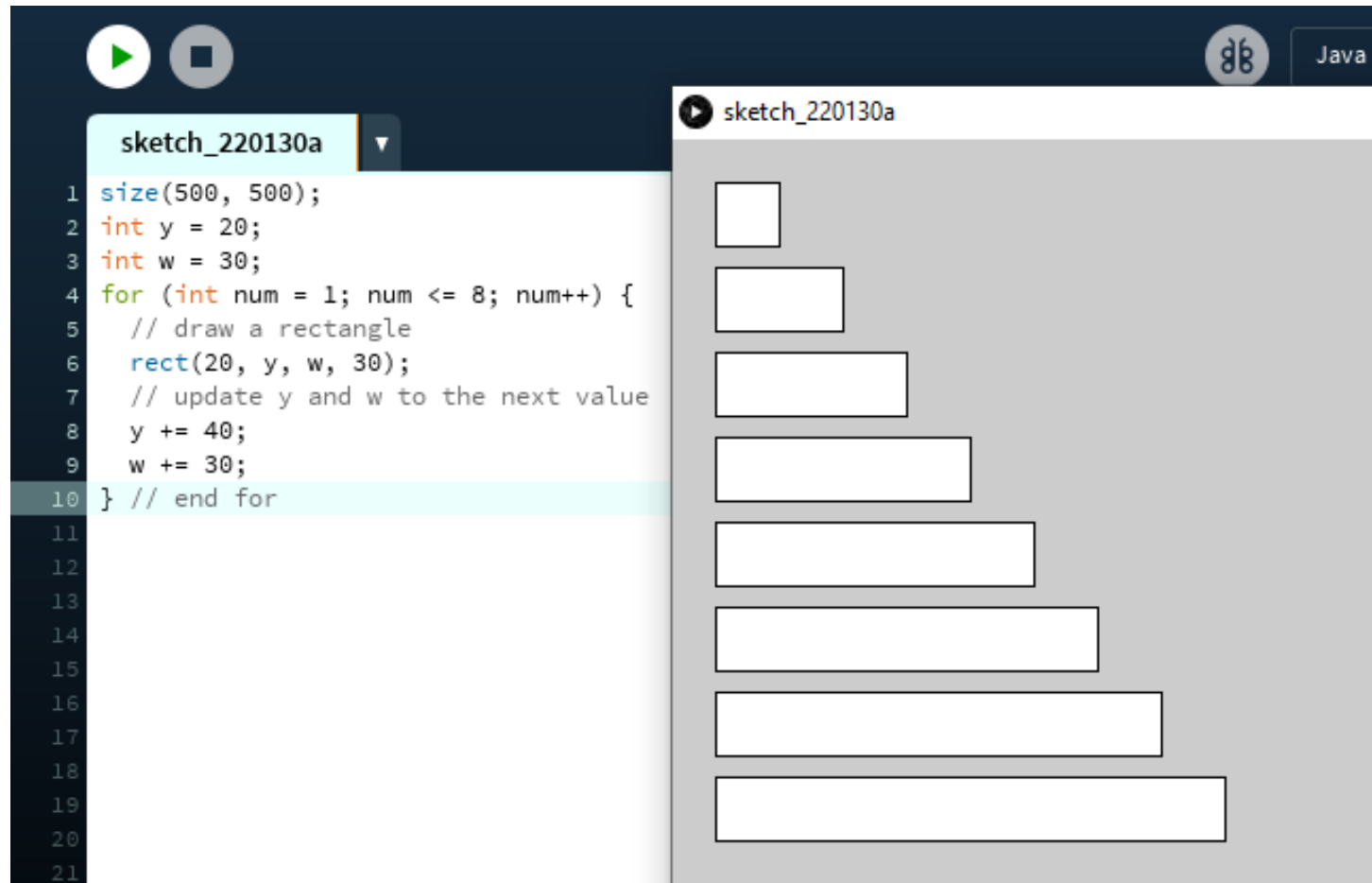
GENERALIZING WITH VARIABLES

- Write the statement *once*, with variables for what changes and constants for everything else
- We can use *y* for the vertical position and *w* for the width of the rectangle

```
rect(20, y, w, 30);
```

- *y* should start at 20 and *w* should start at 30
- Draw the rectangle, then increase *y* by 40 and increase *w* by 30
- Repeat this once for each rectangle to be drawn (in this case, that means 8 times)

HERE'S THE REPETITION!

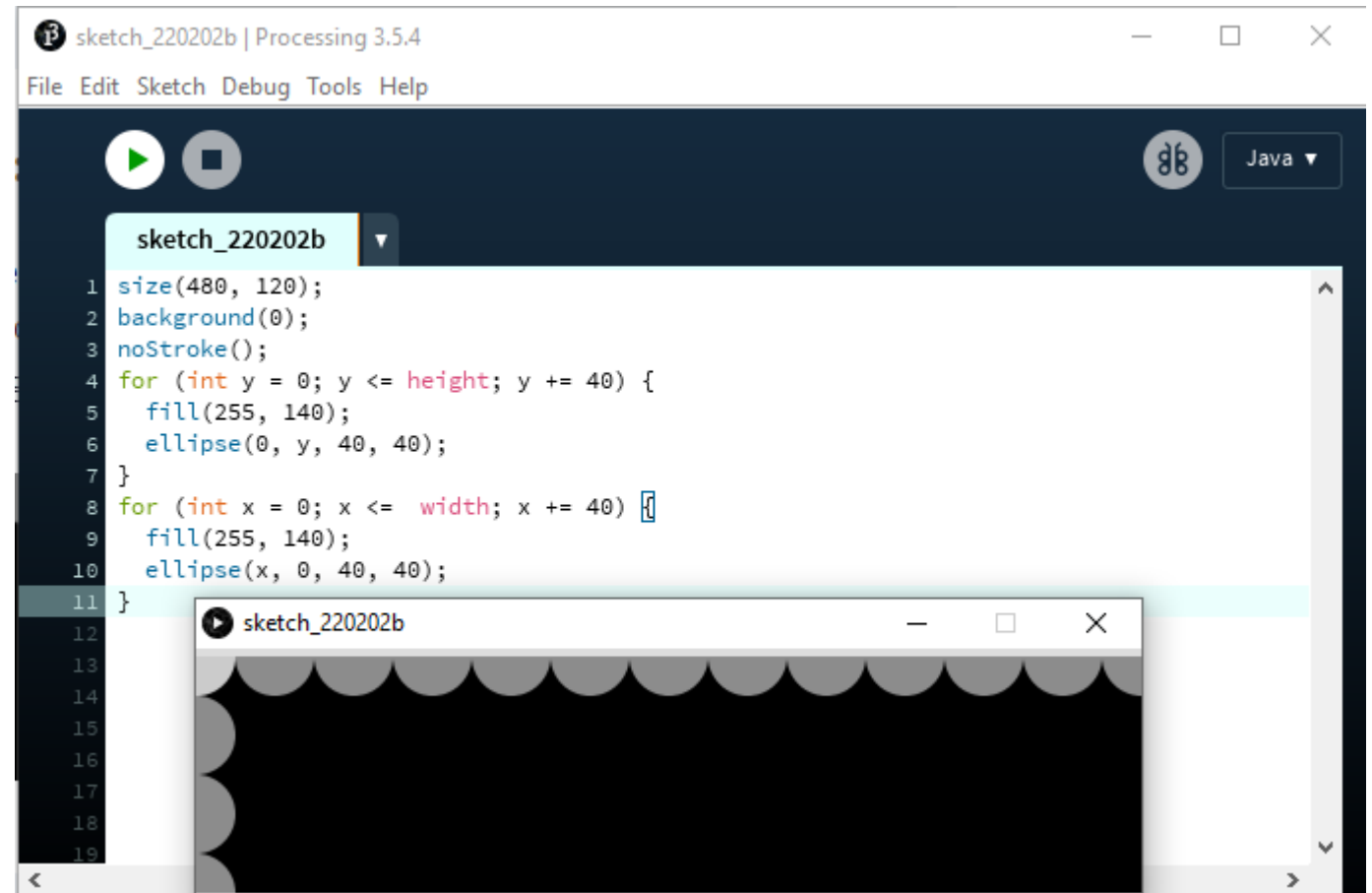


```
1 size(500, 500);
2 int y = 20;
3 int w = 30;
4 for (int num = 1; num <= 8; num++) {
5   // draw a rectangle
6   rect(20, y, w, 30);
7   // update y and w to the next value
8   y += 40;
9   w += 30;
10 } // end for
```

- What if we want to make 5 rectangles, or 12?
- Just change the number of times the loop is executed!

AS SERIES OF for LOOPS

These for loops are adjacent. The vertical circles are drawn by the first loop, and the horizontal circles are drawn by the second loop.



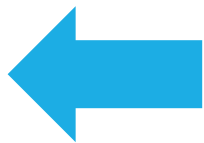
REMEMBER NESTED `if` STATEMENTS?

From Lab 2:

```
int x = 0;

void setup() {
  frameRate(10);
} // end setup

void draw() {
  if (keyPressed) {
    if (key == '+') {
      x++;
    }
    else if (key == '-') {
      x--;
    }
    print("x now = ");
    println(x);
  } // end if (keyPressed)
} // end draw
```



The `if` and `else if` that check for the `+` and `-` keys are nested inside `if (keyPressed)`. That's why the closing curly brace for `if (keyPressed)` comes *after* the closing curly braces for `if (key == '+')` and `if (key == '-')`.

WE CAN EMBED for LOOPS, TOO!

The number of repetitions is multiplied!

- The outer for loop starts
- Then the **entire** inner for loop runs until the condition `x <= width` is false
- Then the outer for loop updates (`y += 40`) and the **entire** inner for loop runs again until the condition `x <= width` is false
- Then the outer for loop updates again. The process repeats until the outer for loop's condition (`y <= height`) is false
- Be careful with syntax! The outer loop's closing curly brace comes after the inner loop's closing curly brace

