12 Repeating Stuff with Loops

Sometimes we need to repeat something over and over. We could just write the same or similar code a bunch of times. This get annoying to type and hard to change.

Instead, we can tell the computer to repeat something for us!

There are two basic kinds of loops: a **While Loop** and a **For Loop**.

The While Loop

Sometime, we need to repeat something until some condition is met.
Here is an example:

```
void setup() {
  size(600, 600);
  background(0);
}

void draw() {
  int x = 0;
  while(x < width) {
   fill(150);
   circle(x, height/2, 16);
   x = x + 20;
  }
}
```

This program draws circles until we reach the end of the screen. This way we don't need to know how many circles we need.

Notice that a while loop is very similar to an **if statement**. The difference is that an if statement runs the code inside once, if true, but a while loop runs its code over and over if the statement is true.

```
if (this is true) {
 do this once;
}

while (this is true) {
 do this over and over;
}
```

While loops are especially useful when we need to keep repeating something until a user does something, like chooses to quit, or looses a game. These are things we can't predict when writing our code, so we need to tell the computer how to act in each situation.

For example, this program changes the number and placement of circles based on the user's mouse position:

```
void setup() {
  size(600, 600);
}

void draw() {
  background(0);
  int x = 0;
  while(x < width) {
    if (mouseX < 1) {
      x = x +1;
    } else {
      x = x + mouseX;
    }
   fill(150);
   circle(x, height/2, 16);
  }
}
```

Notice, that we use a variable to count how many times the loop runs, and we have a condition that will stop the loop. If a loop never stops, it is called an infinite loop. An infinite loop will use all of the computer's memory and crash the program.

For example this program will count infinitely, and print the results to the console (bottom of your coding screen).

```
int x = 2;

void setup() {
  size(600, 600);
    while(true) {
    x = x + x;
    println(x);
  }
}

void draw() {
  background(0);
  circle(width/2, height/2, 50);
}
```

This program will get stuck in the loop, and will never get around to drawing the circle. You will have to manually exit the program with the stop button.

A For Loop

A **For Loop** is a special while loop, that makes it easier to write the most common kind of while loop. It doesn't do anything essentially new, it is just short hand. However, it is far more common than while loops. Code like this is called **syntactic sugar**. It gives programmers a "sweeter" way of writing code. This means the code is shorter or easier, but doesn't really give any new abilities.

A **for loop** is best when we know how many times we want to repeat something—usually for every item in a list.

For example, if we want to make draw 500 stars in random locations, we don't need 500+ lines of code. Instead, this code will generate 500 stars at random locations, with a random size up to 3 pixels.

```
void setup() {
  size(600, 600);
  background(0);
  stroke(255);
  for (int stars = 0; stars < 500; stars++) {
    strokeWeight(random(3));
    point(random(width), random(height));
  }
}
```

To create a for loop use the **for** keyword. In the parenthesis we give three values as input:

- Make a variable to keep count of how many times you loop through your code. Often this variable is called **i** for "index", but you can use whatever name you like.
- Write a condition for when to stop the loop. The loop will continue as long as this condition is **true**, and will stop as soon as this condition is **false**. The simplest example is when your index is higher than a certain value. So `index < 10` will only run until the value of index is 10 or higher.
- Change the value of your index after the code runs. Usually this is simply adding 1 to the index, which is usually written as **i++**. Sometimes, you may want to use a different operation here.

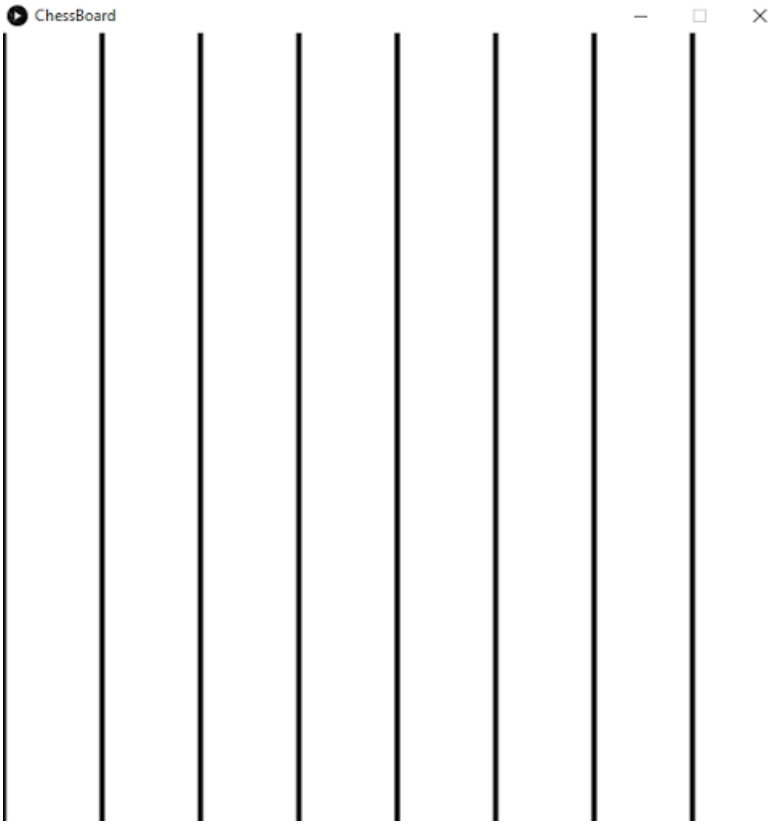
For example, this simple for loop will print the value of **i** to the console 10 times, from 0 to 9.

```
void setup() {
  for (int i = 0; i < 10; i++) {
    println("i is now: " + i);
  }
}
```

Make A Game Board

We can use loops to draw a game board—for example, if we want an 8 x 8 board for chess or checkers. This code will draw 8 lines.

```
void setup() {
  size(600, 600);
  background(255);
  strokeWeight(4);
  for (int lineX = 0; lineX < width; lineX = lineX + width/8) {
    line(lineX, 0, lineX, height);
  }
}
```



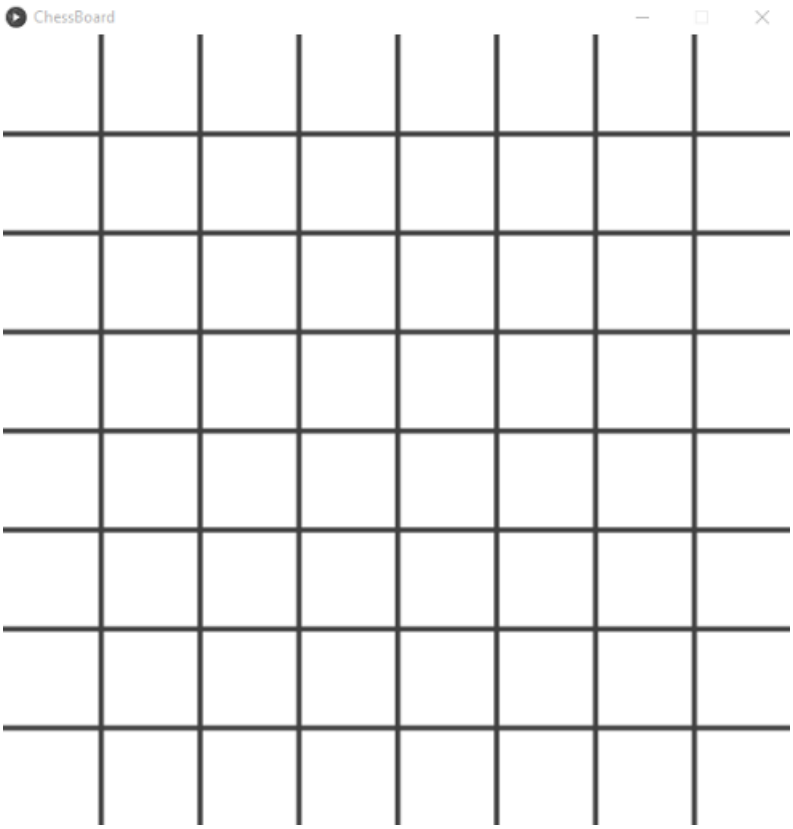
We don't need that first line, so we can start our index variable at a different value. I will make the grid size a variable and use that in our loop.

```
void setup() {
  size(600, 600);
  background(255);
  strokeWeight(4);

  int gridSize = width/8;
  for (int lineX = gridSize; lineX < width; lineX = lineX + gridSize) {
    line(lineX, 0, lineX, height);
  }
}
```

Now we need to add the horizontal lines. The spacing for the **x** and **y** position of the lines is the same—the

gridSize, so we can use the same loop to make both sets of lines. I will rename the variable from lineX to lineSpace.

```
void setup() {
  size(600, 600);
  background(255);
  strokeWeight(4);

  int gridSize = width/8;
  for (int lineSpace = gridSize; lineSpace < width; lineSpace = lineSpace + gridSize) {
    line(lineSpace, 0, lineSpace, height);
    line(0, lineSpace, width, lineSpace);
  }
}
```



This code is also reusable! If we change the gridSize to width/3 it will make a TicTacToe board.

A Chess Board with Squares

We can use a similar process to make a black and white chess board. But for this we need a loop inside of a loop. This is called a **nested loop**.

```
int gridSize = 50;
int margin = 20;
int boardSize = gridSize * 8;

void setup() {
  size (440, 440);
  background (0);

  // Makes a White border around the board
  strokeWeight (3);
  stroke (255);
  noFill();
  square(margin, margin, boardSize);

  // Draw a white square for every other grid position
  noStroke();
  // The first loop goes accross a row from left to right
  for (int x=margin; x<boardSize; x+=gridSize*2) {
  // The second loop goes down a column from top to bottom
    for (int y=margin; y<boardSize; y+=gridSize*2) {
      fill (255);
      square(x, y, gridSize);
      square(x+gridSize, y+gridSize, gridSize);
    }
  }
}
```

Nested loops are confusing, so you can use methods to make things easier to understand. The method names take the place of comments, making our code more organized and easier to understand!

```
int gridSize = 50;
int margin = 20;
int boardSize = gridSize * 8;

void setup() {
  size (440, 440);
  background (0);

  // Makes a White border around the board
  strokeWeight (3);
  stroke (255);
  noFill();
  square(margin, margin, boardSize);

  for (int y=margin; y<boardSize; y+=gridSize*2) {
    drawBoardRow(y);
  }
}

void drawBoardRow(int y) {
  for (int x=margin; x<boardSize; x+=gridSize*2) {
    noStroke();
    fill (255);
    square(x, y, gridSize);
    square(x+gridSize, y+gridSize, gridSize);
  }
}
```

Things To Try

- Write a program that makes 10 circles across the screen.
- Write a program that makes 10 rows of 10 circles.
- Write a program that draws a horizontal gradient. Hint: draw `width` lines, and make each one a darker shade of gray.
- Write a program that makes every pixel in the window a different random color.