MODULE 9: REPETITION, REPETITION, REPETITION

Agenda:

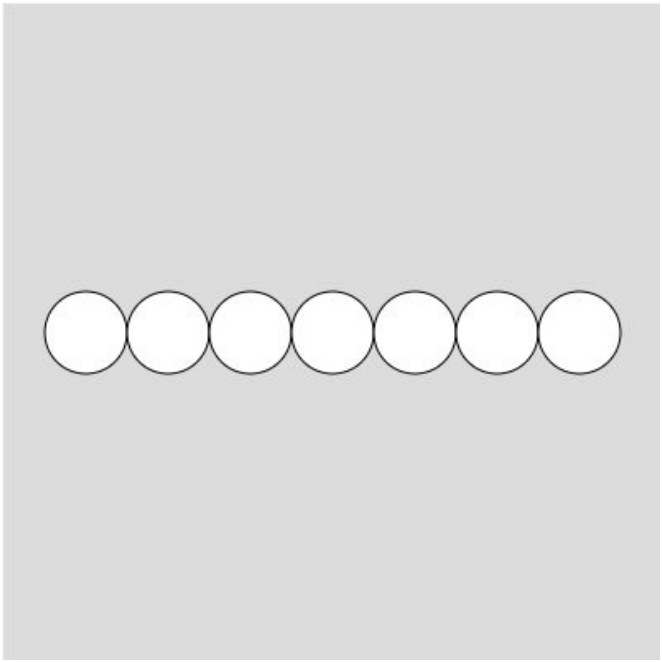
1. Tired of repeating yourself? (10 min)
2. The While loop (10 min)
3. Avoiding infinite loops (10 min)
4. Other useful tools (20 min)

Tired of repeating yourself?

By now, you've probably made sketches that draw the same shape many times. Perhaps you've made a row of ellipses, or a stack of rectangles. You've had to type the ellipse() or rect() function many times. For example, you might have a sketch that looks like this:

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);
  ellipse(50,200,50,50);
  ellipse(100,200,50,50);
  ellipse(150,200,50,50);
  ellipse(200,200,50,50);
  ellipse(250,200,50,50);
  ellipse(300,200,50,50);
  ellipse(350,200,50,50);
}
```



This is fine if you want to draw 6 or 7 shapes. But imagine that you want to have 20 circles, or 100 circles, or even 2000 circles in your sketch. Your fingers would get very tired trying to type the ellipse() function hundreds of times. Luckily, computer programming is designed to make our lives easier --- especially when we want to do repetitive tasks.

In the example above, all of the ellipses are nearly identical. They only difference is their x position. Notice that every x position is 50 pixels larger than the previous one. We can improve our code if we introduce a few variables to replace all of those duplicate hard-coded values. Here's what our code looks like with variables instead:

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);

  var x = 50;
  var y = 200;
  var d = 50;
  var spacing = 50;

  ellipse(x,y,d,d);

  x = x + spacing;
  ellipse(x,y,d,d);

  x = x + spacing;
  ellipse(x,y,d,d);

  x = x + spacing;
  ellipse(x,y,d,d);

  x = x + spacing;
  ellipse(x,y,d,d);

  x = x + spacing;
  ellipse(x,y,d,d);

  x = x + spacing;
  ellipse(x,y,d,d);
}
```

All of our values have been replaced by variables, but we actually had to type even more lines of code to make the same sketch. Every time we want to draw another circle we have to make two lines of code! We want to find an easy way to tell p5.js "Draw this one circle 7 times."

Luckily, every programming language has tools to make this possible. These tools are called “loops,” and they allow us to tell our program how many times it should repeat a command.

The While loop

The first loop we will learn is called the “while” loop. The while loop (and all loops) work just like the “if/else” statements we learned last module. We use a boolean test (asking the program if something is true or false). With the “if” statement, we told the program to execute some commands if the statement is true:

```
if( true ) {
  //do the commands inside this block of code
}
```

With “if” and “else” to program will execute the instructions inside the block of code one time and then move on with the rest of the program. When we use a “while” loop, the commands inside the block of code will be repeated over and over until the boolean test becomes false:

```
while(myVariable < 10) {
  //repeat the commands inside this block of code until myVariable is equal to 10 or more
}
```

As long as myVariable is less than 10, the program will repeat whatever commands you’ve written inside the block of code.

Now let’s try to use this new tool in our example. We want the program to draw ellipses until the value of x becomes greater than 350. So, if we want to put this into a while loop, it looks like this:

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);

  var x = 50;
  var y = 200;
  var d = 50;
  var spacing = 50;
```

```
  while(x <= 350) {
    ellipse(x,y,d,d);
    x = x + spacing;
  }
}
```

We just saved ourselves a lot of typing by using a while loop. Instead of writing the ellipse() function every time we want to draw an ellipse, we use the while loop to say “as long as x is less than or equal to 350, draw an ellipse and also increase the value of x.”

The best part of using all these variables and a while loop is that it becomes very easy to draw as many circles as we want. Try changing the the variable d and the variable spacing to 20 and press Play.

3 rules for loops

In general, there are three important things to ask yourself when you’re using a loop.

1. **What is the initial condition for your loop?**
2. **When should your loop stop?**
3. **What do you want the loop to do?**

Let’s think through these 3 steps with another example. Let’s say we want to draw horizontal lines across our canvas. These lines will extend from 0 on the x-axis to the width of the canvas. So it will look something like this:

```
line(0,y,width,y);
```

If we start at the very top of our canvas we want our initial condition (our first value of y) to be zero. So we have to add this line of code to our sketch.

```
var y = 0;                          //STEP 1 - the initial condition for our loop
```

Now when should our loop stop? If we want to draw horizontal lines on our whole canvas, we want to stop drawing lines once we reach the full height of our canvas. So our loop should stop when y is greater than height.

```
while(y < height)                   //STEP 2 - our loop should stop when y is > height
```

We answer, the first two questions. Now we just have to decide what we want our loop to do. In other words, what commands do we put inside our while loop? We know we want to draw a line. We also want to move down the canvas some amount so we can draw more lines.

```
line(0,y,width,y);                    //STEP 3 - what we want the loop to do
y = y + 10;
```

If we put these three steps together, we have code like this:

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(0);

  var y = 0;

  while(y<height) {
    line(0,y,width,y);
    y = y + 10;
  }
}
```

If you want more lines on your canvas, just decrease how much you add to y each time. If you want fewer lines, make y increase by a greater amount each time.

Avoiding infinite loops

Hopefully you're starting to see the power of loops. They can make your life much easier and your code much less messy. But, as the saying goes, "with great power comes great responsibility." Loops are very powerful, but that also makes them a bit dangerous. If we don't use them properly we can break our code and sometimes cause even bigger problems on our computer.

Whenever we use loops, we have to make sure we avoid something called "infinite loops." An infinite loop happens when the code inside a while loop starts executing and it never stops. If the thing we are testing in our while loop (the boolean test) is always true, the loop will never stop. Consider the following:

```
while(7 == 7) {
  //do the code inside of this block
}
```

The condition we are testing, “Is 7 equal to 7?”, is always true! Remember the while loop only stops when the condition becomes false. So this while loop will repeat forever, which is bad! Our program won’t work unless we have something called an “exit condition”.

An exit condition simply means that at some point the while loop will stop looping and move on with the rest of your program. Consider this example:

```
var d = 10;

while(d < 100) {
  ellipse(200,200,d,d);
  d = d - 1;
}
```

Because we’re decreasing the value of d every time we run through the loop, the “exit condition” will never be met. Our variable d will *always* be less than 100. This will result in an infinite loop and make your program crash.

So be very careful when you’re using while loops. Think through the 3 rules and make sure that whatever you put inside the while( ) parenthesis will eventually be false, so the loop will stop running and move on to the rest of your program.

Other useful tools

The p5.js language has so many tools and functions available to make interesting drawings, animations and interaction. The full list of tools is available at p5js.org/reference
You should take a moment to look through some of these tools on your own. If you are ever confused about a function you’ve learned or don’t remember exactly how it works, the Reference provides examples and explanations. For now, we’ll introduce two more to make your sketches more interesting and controllable.

random()

The random() function is a random number generator. If you’re tired of having your sketches be predictable and always look the same, the random() function is the tool you need. The random() takes 0, 1, or 2 arguments and it returns a number that you can store in a variable. Let’s go through three examples.

```
var myVariable = random();                  //1st example
var mySecondVariable = random(15);          //2nd example
```

var myThirdVariable = random(50,100); **//3rd example**

When you use the random number function it's best to store your number in a variable you create.
The 1st example will give you a random number between 0 and 1 - like 0.433264 or 0.7239.

The 2nd example returns a random number between 0 and 15 (or whatever number you put inside the parenthesis) - like 2.452546 or 14.9965

The 3rd example returns a random number between 50 and 100 (or whatever numbers you choose for the minimum and maximum range of a random number you want).

Let's try this out in a sketch:

```
function setup() {
  createCanvas(400, 400);
  background(220);
}

function draw() {
  var x = random(width);
  var y = random(height);
  ellipse(x,y,20,20);
}
```

Every time we loop through draw, we choose a new random x position (between 0 and the width of our canvas), and a new random y position between 0 and the height, then it draws an ellipse to the canvas.

You can make this more interesting by giving random values for the fill color and for the size of the ellipse. What if our draw loop was like this instead?

```
function draw() {
  var x = random(width);
  var y = random(height);
  var diam = random(15,25);
  var r = random(255);
  var g = random(10,100);
  var b = random(150);

  fill(r,g,b);
  ellipse(x,y,diam,diam);
}
```

Press play and watch your canvas fill up with pretty circles. Remember that you can adjust the numbers inside the random() function to get different size circles and different colors. What if you wanted to only have random circles in the top left quadrant of your canvas?

constrain()

Another useful tool is the constrain() function. As the name suggest, this function constrains or limits the values of a variable. It works like this:

```
var newVariable = constrain(x,100,200);
```

The constrain() function takes three values or arguments. The first is the variable you want to constrain, the 2nd is the minimum value for that variable, and the 3rd is the maximum value you want for that variable. This code is the equivalent of saying “I want to constrain my variable x between the numbers 100 and 200.” This function is especially useful for controlling your colors.

Enter the following code into the p5.js editor and see what happens when you move your mouse from the left half to the right half of the canvas.

```
var col = 0;
function setup() {
  createCanvas(400, 400);
  background(220);
}

function draw() {
  background(col);
  stroke(0);
  line(width/2,0,width/2,height);

  if(mouseX > width/2) {
    col = col + 1;
  } else {
    col = col - 1;
  }
  col = constrain(col,0,255);
}
```

Notice how the background color gets lighter when the mouse is on the right side of the canvas? Using an if/else statement we are adding or subtracting from our variable col (which

controls the background color). The last line of our code makes sure that the value of “col” never becomes less than 0 or greater than 255.

Try remaking this sketch with 3 r, g, b values instead of just one value controlling the background color.

Exercise: Make many characters in one sketch

For this exercise, create a simple design, face, or character. Then use a while() loop to make multiple copies of that character.

Incorporate the random() and constrain() functions to make each version of the character slightly different. Maybe each version will have different size eyes or different colors for the face. Think of all the different parameters you could control or make random to give lots of variety to your design.

Notice in this example, all the code is written in the setup() function. What happens if you move it to the draw loop?

```
function setup() {
  createCanvas(400, 400);

    background(220);

  var x = 50;
  while(x < width) {
    var r = random(255);
    var g = random(255);
    var b = random(255);
    var eyeSize = random(10,25);

    fill(r,g,b);
    ellipse(x,200,100,100); //face


    //eyes
    fill(255-r,255-g,255-b);
    ellipse(x-20,180,eyeSize,eyeSize);
    ellipse(x+20,180,eyeSize,eyeSize);
```

```
    //mouth
    fill(g,b,r);
    rect(x-20,210,40,random(5,15));

            x = x + 100;
  }
}
```