Note: there is no included solution program for lesson 8.

In this lesson we're going to do a little bit more non-Romi practice in order to familiarize you with some concepts that are critically important, but that you won't see too much of in the Romi code. Specifically, we're going to talk about loops. A *loop* is a chunk of code that is designed to run multiple times. You may have already realized that this is happening in your Romi code, because the program runs indefinitely until you turn the robot off. The loops in the Romi code are behind the scenes a little bit, so not as visible to you. We will talk about where they exist and how they result in your code running indefinitely. We'll also do some desktop practice without the Romi. The next two lessons will be review and practice, with heavy focus on Romi projects as opposed to new code concepts.

Loops

Start by watching this video, and follow along in a scratch project in VSCode: https://www.youtube.com/watch?v=6djggrlkHY8

Once you're done with that, open a fresh Romi project. Open Robot.java. This file contains a number of methods that essentially are run in loops. Look at line 85, which is a method declaration for teleopPeriodic(). This function is called repeatedly while teleop is enabled. Right now there's nothing in here, but if you were to, for example, put a SOP in there and enable teleop, you would see it start outputting to the terminal. You can see the same thing for disabled, autonomous, and test modes on lines 53, 70, and 95, respectively. There's also a method called robotPeriodic declared on line 40. This one is similar but it runs all the time regardless of which mode the robot is in. Unlike the other methods, you'll notice this one does have a line of code in it – line 45, CommandScheduler.getInstance().run();. You've created and used some commands in prior lessons, and this line is basically what makes them all work, since robotPeriodic is running in a loop behind the scenes. When line 45 runs, an object called the CommandScheduler basically looks at all the commands that need to do things and makes them do those things. You'll learn more about commands as we keep working with Romi code, but this is where the magic happens. Although you don't see the words "for" or "while" in these examples, the concepts are the same.

Now let's do some practice writing your own loops. Here are some exercises to complete in a desktop VSCode project. They'll give you practice using loops as well as reinforce your previously developed skills. A quick note – why are we doing desktop projects for loops instead of Romi code? Well, the simple answer is that if you mis-use loops in your robot code, you can create some dangerous situations. You just saw that the command scheduler needs to run constantly in robotPeriodic. If you have a loop in your robot code that doesn't terminate quickly, not only can the commandScheduler not run, but neither can any code that is not in your loop. That might include critical safety features like telling your motors to turn off. If your motors are running full power and the code turning them off can't run, your robot will not be able to stop. There *are* situations where loops are good in robot code, but they're a bit fewer, further between, and often have to do with more advanced concepts. This, combined with the fact that your code is already set up to effectively be self-looping because of your commands running all the time and each mode of the robot having a period method where you can add code that you want to run over and over, means you don't need as many loops in your robot code. For desktop applications, on the other hand, if you want your code to run over and over you'll have to create that functionality yourself, meaning you'll need a lot of loops. With that being said, here are the exercises. In lesson 4 you learned how to use the Scanner to generate user input. You'll need to use the

Scanner again here. To complete these exercises, make one program, and one method for each exercise. In your main method, call each exercise's method, so your single completed program does all of the exercises.

1. Write a for loop that SOP's every integer from 0 to 9.
2. Write a while loop that outputs every integer from -10 to positive 20.
3. Take an integer input from the user and compute the "sum of squares" from 1 to that number. For example, if the user entered 10, the answer would be 1^2 + 2^2 + ... + 10^2. Output the answer. To check your work - if you enter 10, the answer should be 385. If you enter 11, it should be 506. You'll need to declare a variable outside of your loop to keep track of the sum so far.
4. Take an integer input and compute the "square of sum" from 1 to that number. So if they entered 10, it would be (1 + 2 + ... + 10)^2. Output the answer. To check: 10 comes out to 3025, and 11 comes out to 4356.
5. Now calculate the "sum-square difference" - that is, the difference between the square of sum and the sum of squares. So basically, the result of problem 2 minus the result of problem 1. If you enter 100, the answer should be 25164150.
6. Output a multiplication table for 1-10. Note that in order to do this you will need to do something a bit new - a "nested" loop. You'll have one for loop that goes from 1 to 10, and then you'll have another for loop that also goes from 1 to 10 *inside* the first for loop. Your nested for loop will complete fully for every cycle of the outer loop. Remember that you can use System.out.print instead of .println to output something without a new line, and you can do System.out.println(""); to output a new line and nothing else. So your result should look something like the following (the exact spacing of the numbers isn't too important.)
   1 2 3 4 5 6 7 8 9 10
   2 4 6 8 10 12 14 16 18 20
   3 6 9 12 15 18 21 24 27 30
   .....
   10 20 30 40 50 60 70 80 90 100