You've made it through 19 lessons of Romi programming and there's only one lesson to go! In this lesson, we'll combine a lot of the concepts you've learned over the last few months into a single project. There isn't much new content to learn here, but it will take you some time to piece together all the different pieces of the project from prior projects and make them work seamlessly together.

Issuing Commands in Teleop

You learned how to bind a command to a button on the joystick in teleoperated mode in lesson 3, but the commands we used there acted on the on-board IO subsystem and not the drivetrain. One capability you have that can be useful is to bind commands that *act upon your drivetrain* to buttons in teleop. This lets your robot automatically perform actions for you, instead of forcing you as a driver to do them yourself. If you do this, while the command is running, you will lose control of the robot, because as soon as your command starts running, it will interrupt the default command that the drivetrain is running. When your command finishes, the drivetrain will resume running its default command, which is your drive code, and you'll re-gain control. As a general rule this is not a problem, but you do need to be careful not to issue any commands that take longer than expected or don't end properly, or you could lose control of your robot. This also means you need to be ready to disable your robot all the time while testing so it doesn't accidentally drive away from you or fall off your desk onto the floor.

In lesson 3 you used the "whenActive" and "whenInactive" methods of the JoystickButton class to bind commands to run while the button was held down, or not held down. Sometimes you want commands to run a single time when the button is pressed. For that, there is the "whenPressed" method. Here's an example of using the whenPressed method to bind the built-in TurnDegrees command to a button press in RobotContainer:

```
JoystickButton button1 = new JoystickButton(m_controller, 1);
button1.whenPressed(new TurnDegrees(-0.5, 180, m_drivetrain));
```

This code will make the robot turn 180 degrees whenever you press the button. You will not be able to drive the robot while it is turning but will regain control as soon as it finishes. You can test this out for yourself in a RomiReference project by copying the code above (you'll need to import the TurnDegrees command using the lightbulb.) Some examples of useful commands you might bind to buttons for teleop would be to turn left or right a fixed number of degrees, or to turn to a known orientation. Say you were completing a timed challenge that involved a series of 45 degree turns. You could certainly complete the challenge manually by controlling the robot, but if you bound a button to turning 45 degrees, you could complete it faster because your code would automatically turn the right amount. Another example might be driving a fixed distance – say you were supposed to move back and forth between two points exactly 20 inches apart. Why force yourself to do all the work when you could just bind that to a button? As you spend more time writing code and playing with robots, you'll find more examples of things you can automate for yourself.

Project – Automating Teleop

For the final project of the course, you'll combine what you've learned and coded in prior projects into a single program. Build off your code from projects 16-18, and create a program that does the following:

- Has four different functions bound to buttons in teleoperated mode:

    - One turns the robot left 90 degrees
    - One turns the robot right 90 degrees
    - One turns the robot around (180 degrees)
    - One returns the robot to its starting orientation (the orientation it was in when the program began running.)
- Uses your custom arcadeDrive code to drive while *not* executing one of the functions in the prior bullet point.
- Has drive straight code implemented and passes the "tray test" where you rotate the robot around and it maintains its orientation.
- Does *not* automatically revert to a prior orientation after you execute one of the button press commands. That means you will need to update the target heading while these commands are running, or at the very least when they finish running.
- Successfully passes off control between the default drive command and the button-issued commands. You can test this by driving the robot forward at a slow speed, confirming that it's behaving as expected, and then issuing a button command *but still attempting to drive forward at slow speed while the button-issued command is running.* The robot should complete the button-issued command instead of driving forward, and then immediately resume driving forward after the button-issued command is completed.

Once you've implemented and tested all these features, congratulations, you have finished the course! You've gained a lot of knowledge and developed some strong skills. This course is meant to be a challenge, so even if you found these projects difficult, do not worry – as you continue to gain practice, you will continue to increase your understanding, you'll be able to write code faster, and you'll gain confidence in your abilities. If you enjoyed working with the Romi, the next step for you would be to find a local FIRST team at your school or in your community competing in either FIRST Robotics Competition or FIRST Tech Challenge. Another step would be to continue learning Java and to take the AP Computer Science exam to get college credit. Software programming takes time to learn, but you are well on your way and if you continue your learning you will find it to be a challenging, profitable, and most importantly fulfilling skill to master.