Robot Mesh Curriculum

User Manual

Created: Tuesday, June 05, 2018

Robot Mesh Curriculum

Table of Contents

Robot Mesh VEX IQ Curriculum Guide

Introduction

Students of today are the technology leaders of the future, using engineering approaches to solve problems and meet the needs of people. The VEX IQ robotics platform is widely used in education, after-school activities, and even in homes as a great way to engage young people in the STEM fields of science, technology, education and mathematics. You can build amazing robots using VEX IQ, and you can use Robot Mesh Studio™ to make them work.
Congratulations on choosing Robot Mesh Studio (RM Studio) to program your VEX robots. RM Studio is the first development suite for VEX that incorporates Flowol™, Blockly and Python software development environments, with Robot Mesh 3D Mimic™ online CAD and cloud-based programming. The focus of this curriculum is Robot Mesh Blockly for VEX IQ, an interactive graphical software development environment which builds native Python code for the robot. The tight integration between Blockly and Python provides a solution for educators where advanced students can be working in Python in the same software that emerging learners are using Blockly.

Using This Activity Guide

These activities will walk a beginner through basic programming structures, the use of VEX IQ parts and sensors, and give the student a chance to learn about programming robots. They are hands-on, with a lot of activity and not much book time. Each activity includes an opportunity for the student to respond to the material, and most include extension activities to allow the students to dive deeper into the activity's subject area. Most students will want to start at the beginning, and progress through the activities in order.
The earlier activities show sample Blockly code and encourage the student to build, test, and evaluate. The last few activities do not include sample code, as they should be doing their own coding at this point.
Blockly has very little arbitrary syntax, so we can focus on concepts, and not the minutiae of placing punctuation. The learning curve is rapid, and each step builds on previous learning.
Every activity encourages the students to look at the Python code that is build during Blockly programming. We want them to understand that Blockly is a great way to get started, but there is also a seamless path to an industry-standard, modern programming language with even more power and flexibility. The student can start in grade 4 with Blockly and keep programming with Python through high

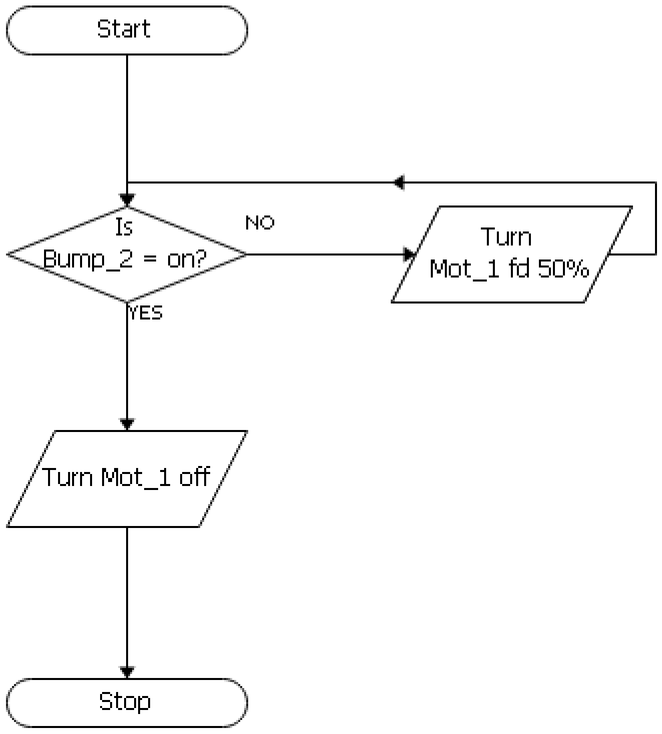
school, college and into the professional world. Since Blockly and Python both work on VEX EDR, too, there is an easy path to the VEX platform for more-experienced students. No matter the platform or language, Robot Mesh Studio is the best solution to educational robotics programming.

Programming with Robot Mesh Studio

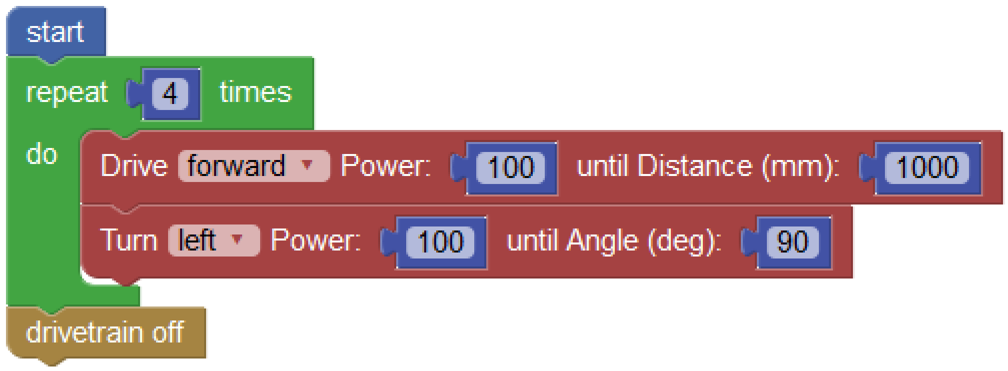
Robot Mesh Studio provides four types of programming for VEX IQ robots for teachers to use with their classes.

Controller Express is designed to get kids programming quickly by automatically generating code for simple VEX IQ Controller programming. Students configure the motors and joystick, then start driving the robot in a matter of minutes. This is the simplest way to get started with programming.

Flowol allows students of all ages to develop logical reasoning and problem solving talents, develop programming skills and explore the world of automatic, autonomous systems and robots. Programming visually with a flowchart allows the student to focus on the logic of their solution rather than the syntax of a written program.

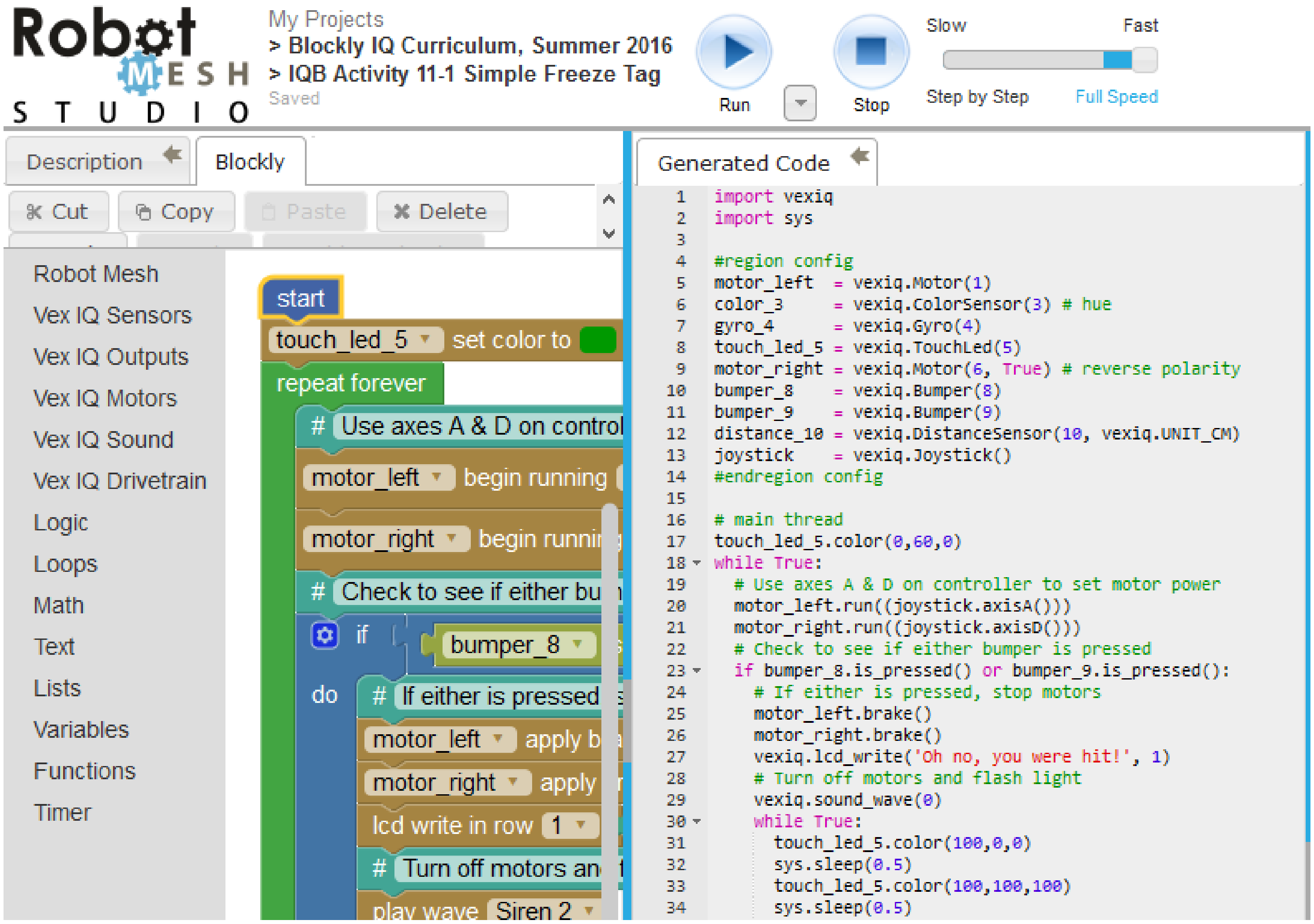


Blockly is a visual block programming editor, which creates Python code that is then downloaded to the robot for execution. Blockly is an open-source project of Google and runs in a web browser, so that schools do not have to download and install software on PCs.



```
start
repeat 4 times
do
    Drive forward Power: 100 until Distance (mm): 1000
    Turn left Power: 100 until Angle (deg): 90
drivetrain off
```

Python is a modern high-level dynamic programming language, which emphasizes code readability, and is generally easier to read and write than languages like C or Java. The language is widely used in all aspects of programming, and is especially popular in science and education. Python is a procedural language, and supports functional, object-oriented, and imperative programming, and uses dynamic data types and automatic memory management. Blockly programs generate Python code as shown below.



```
import vexiq
import sys

#region config
motor_left  = vexiq.Motor(1)
color_3     = vexiq.ColorSensor(3) # hue
gyro_4      = vexiq.Gyro(4)
touch_led_5 = vexiq.TouchLed(5)
motor_right = vexiq.Motor(6, True) # reverse polarity
bumper_8    = vexiq.Bumper(8)
bumper_9    = vexiq.Bumper(9)
distance_10 = vexiq.DistanceSensor(10, vexiq.UNIT_CM)
joystick    = vexiq.Joystick()
#endregion config

# main thread
touch_led_5.color(0,60,0)
while True:
  # Use axes A & D on controller to set motor power
  motor_left.run((joystick.axisA()))
  motor_right.run((joystick.axisD()))
  # Check to see if either bumper is pressed
  if bumper_8.is_pressed() or bumper_9.is_pressed():
    # If either is pressed, stop motors
    motor_left.brake()
    motor_right.brake()
    vexiq.lcd_write('Oh no, you were hit!', 1)
    # Turn off motors and flash light
    vexiq.sound_wave(0)
    while True:
      touch_led_5.color(100,0,0)
      sys.sleep(0.5)
      touch_led_5.color(100,100,100)
      sys.sleep(0.5)
```

All of these options are built into Robot Mesh Studio, so you can have your students start with Controller Express and then progress to Flowol, Blockly and Python when your students are ready. Controller

Express rapidly gives them a moving robot, Flowol teaches them logic, Blockly allows simple graphical programming, and Python exposes the full suite of powerful programming tools.
Python in Robot Mesh Studio

Controller Express, Flowol, Blockly and the Python IDE all generate Python code which is then downloaded to the robot. This means that you can develop projects in Controller Express, Flowol and Blockly, then copy them into the Python editor for further programming.

Works with VEX IQ Curriculum

Use this curriculum with the mechatronic curriculum in the "VEX IQ Robotics Education Guide" and the robots built in the "VEX IQ Build Instructions" to combine robot building with programming. The VEX IQ Robotics Education Guide is available for $9.99 at the Robot Mesh website, or you can use it online at the VEX website (search on "VEX IQ Curriculum"). The VEX IQ Build Instructions are included with VEX IQ kits.

Activities

The RM Studio curriculum is hands-on activity-based learning, with the content divided into activities, each of which includes an Activity Guide. Each activity includes:

1. The activity description, including step-by-step instructions
2. Instructions for students
3. Programming language "how to" information
4. Any other content students need for understandingPossible activity extension ideas to enhance the experience
5. Post-activity student check-up

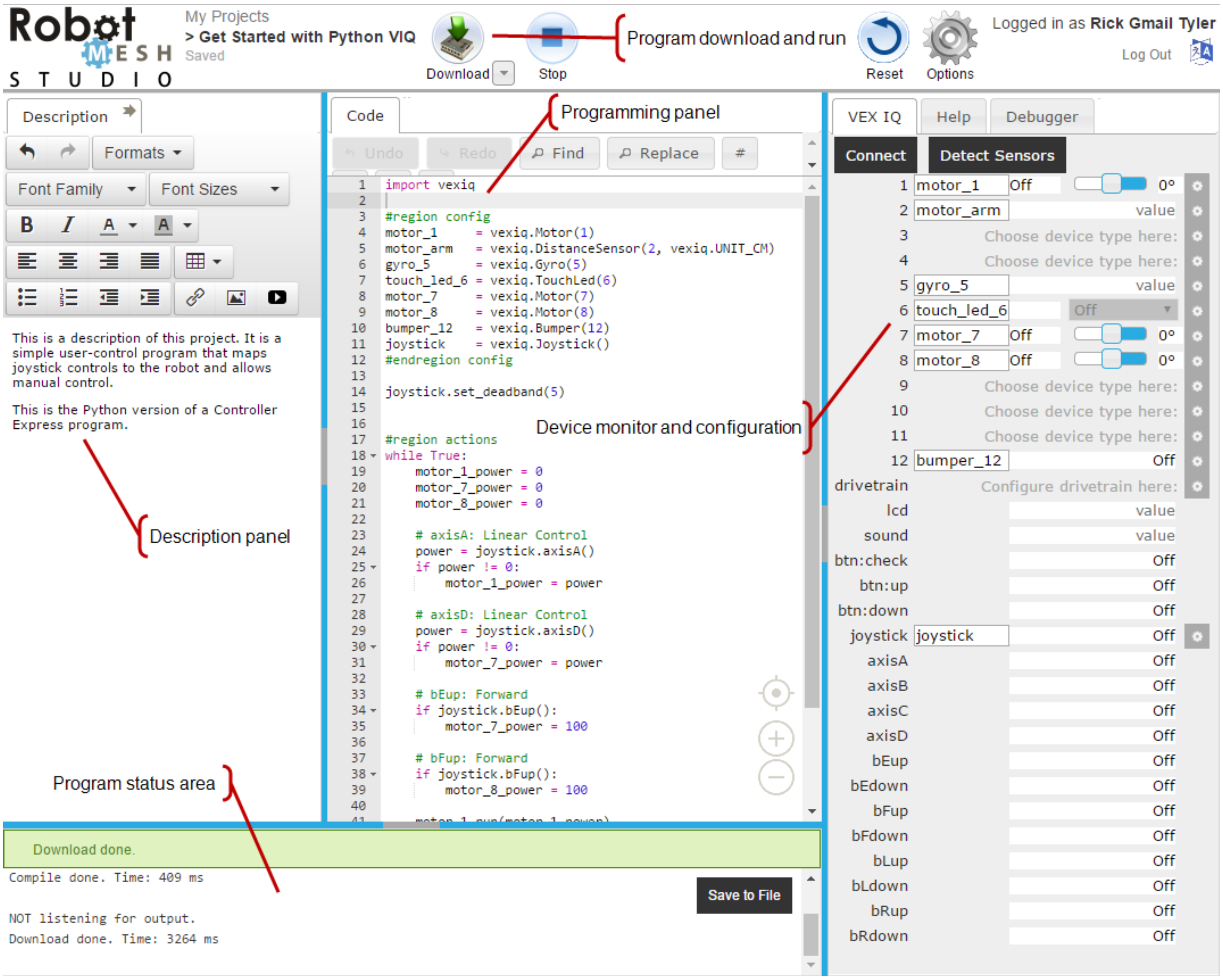
Get Started with Python

Robotics Programming with Python

Python is a modern high-level multipurpose programming language. It was designed to make writing code simpler without losing the power of a low-level language like C. Robot Mesh has extended Python to be the premier VEX language for both VEX EDR and VEX IQ.
This block, for example, tells the robot's computer to set the left motor to a power level equal to joystick Channel A from the remote controller:

```
# axisA: Linear Control
power = joystick.axisA()
if power != 0:
motor_1_power = power
```

The RM Studio development environment looks like this:



This is a description of this project. It is a simple user-control program that maps joystick controls to the robot and allows manual control.

This is the Python version of a Controller Express program.

```
import vexiq

#region config
motor_1     = vexiq.Motor(1)
motor_arm   = vexiq.DistanceSensor(2, vexiq.UNIT_CM)
gyro_5      = vexiq.Gyro(5)
touch_led_6 = vexiq.TouchLed(6)
motor_7     = vexiq.Motor(7)
motor_8     = vexiq.Motor(8)
bumper_12   = vexiq.Bumper(12)
joystick    = vexiq.Joystick()
#endregion config

joystick.set_deadband(5)

#region actions
while True:
    motor_1_power = 0
    motor_7_power = 0
    motor_8_power = 0

    # axisA: Linear Control
    power = joystick.axisA()
    if power != 0:
        motor_1_power = power

    # axisD: Linear Control
    power = joystick.axisD()
    if power != 0:
        motor_7_power = power

    # bEup: Forward
    if joystick.bEup():
        motor_7_power = 100

    # bFup: Forward
    if joystick.bFup():
        motor_8_power = 100

```

1	motor_1	Off 0°
2	motor_arm	value
3		Choose device type here:
4		Choose device type here:
5	gyro_5	value
6	touch_led_6	Off
7	motor_7	Off 0°
8	motor_8	Off 0°
9		Choose device type here:
10		Choose device type here:
11		Choose device type here:
12	bumper_12	Off
drivetrain		Configure drivetrain here:
lcd		value
sound		value
btn:check		Off
btn:up		Off
btn:down		Off
joystick	joystick	Off
axisA		Off
axisB		Off
axisC		Off
axisD		Off
bEup		Off
bEdown		Off
bFup		Off
bFdown		Off
bLup		Off
bLdown		Off
bRup		Off
bRdown		Off

```
Download done.
Compile done. Time: 409 ms

NOT listening for output.
Download done. Time: 3264 ms
```

The development environment is divided into five zones:

1. The Project Download is where you manage your project and download it to your robot for execution and test. This interface also controls the interactive debugger.
2. The description panel is a place where you can paste or enter information about your project.
3. Write Python code in the Programming Area, which is where you build your program.
4. The Device Monitor and Configuration area is where you can see what is plugged into the VEX IQ Brain, name them, and set up configuration options for controllers, motors, drivetrains and sensors.
5. The Program Status area is where you can see compiler notes, download status, and program status messages from the Print command.

Python is meant to be simple enough for beginners, but include enough Python functionality to be useful for experienced programmers. The first activity uses the Controller Express option to build a simple program to allow manual control of the basic robot. These activities develop the basic programming skills students need to write their own programs and for their own robots.

2. Controller Express Activity

Activity Overview

Controller Express is the fastest way to get started with VEX IQ programming. Using a friendly graphical interface, you will build a robot, click on some options, and then be driving your robot in short order. In learning to use Controller Express, you will also will learn about the VEX IQ Brain and motors.
We will create a "tank drive" control for your first robot. Pushing up on the left joystick (the "A" channel) will make the left side go forward, and on the right joystick (the "D" channel) will make the right side go forward. Pull them both down to go backwards, and move them in opposite directions to turn. In this mode, we will not be using the side-to-side motion of the joysticks, so there will be no actions on channels B or C.

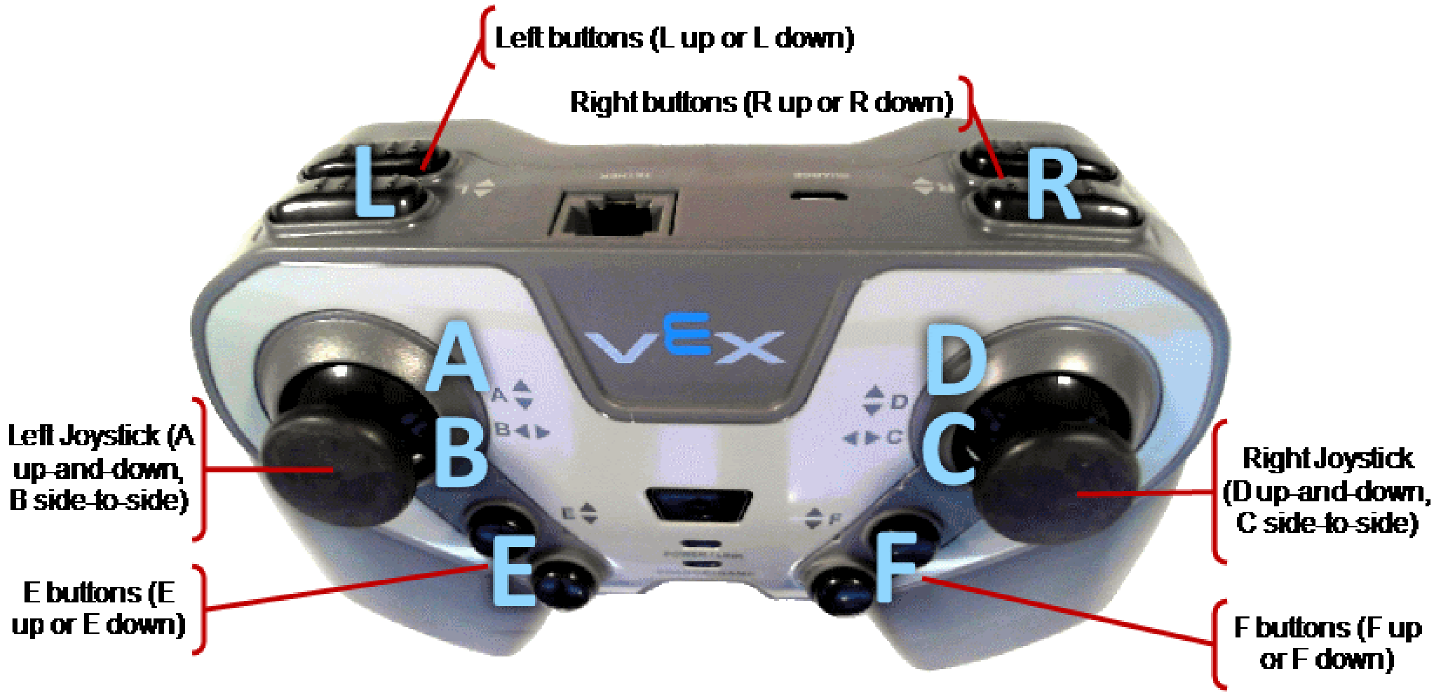
Activity Preparation

Understanding the VEX IQ Controller

The Controller has eight controls that send signals to the Brain. These are:

- Left and right joysticks (each joystick has vertical and horizontal controls, for a total of two control channels for each of the two joysticks mounted on the controller)
- The buttons under the joysticks (1 control each)
- The "index finger" buttons on the side of the controller (1 control each)

Here is what these look like on the controller:

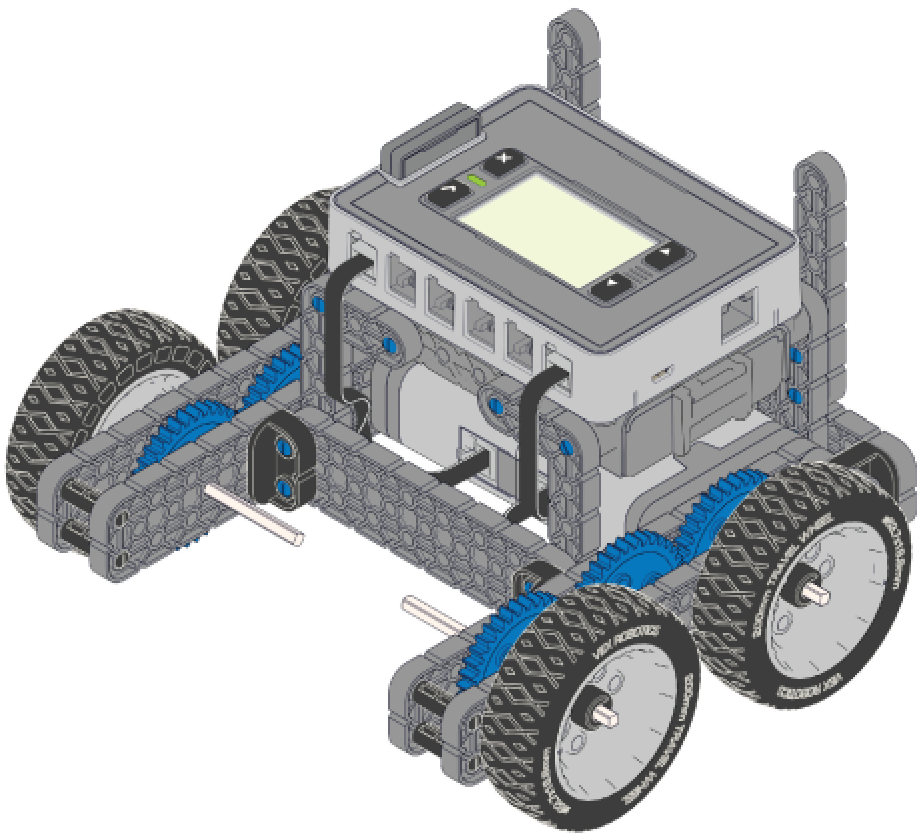

Left buttons (L up or L down)
Right buttons (R up or R down)
L
R
A
B
D
C
E
F
Left Joystick (A up-and-down, B side-to-side)
Right Joystick (D up-and-down, C side-to-side)
E buttons (E up or E down)
F buttons (F up or F down)

When your Brain is programmed correctly, it will see the inputs from each of these eight controls and then run the program you wrote for it.

Controller Inputs to Brain

The input from the two joysticks is a number, ranging from -100 to 100. The buttons, though, only have two values; and the Controller sends either "up" or "down" to the Brain, depending on which button you pressed. The joysticks are called "analog" sensors, and the buttons are "digital." This will make more sense when we introduce more sensors for your robot.

Start by building the Standard Drive Base Robot in the VEX IQ Build Instructions, steps 1-19. After you build your robot, we can get started.

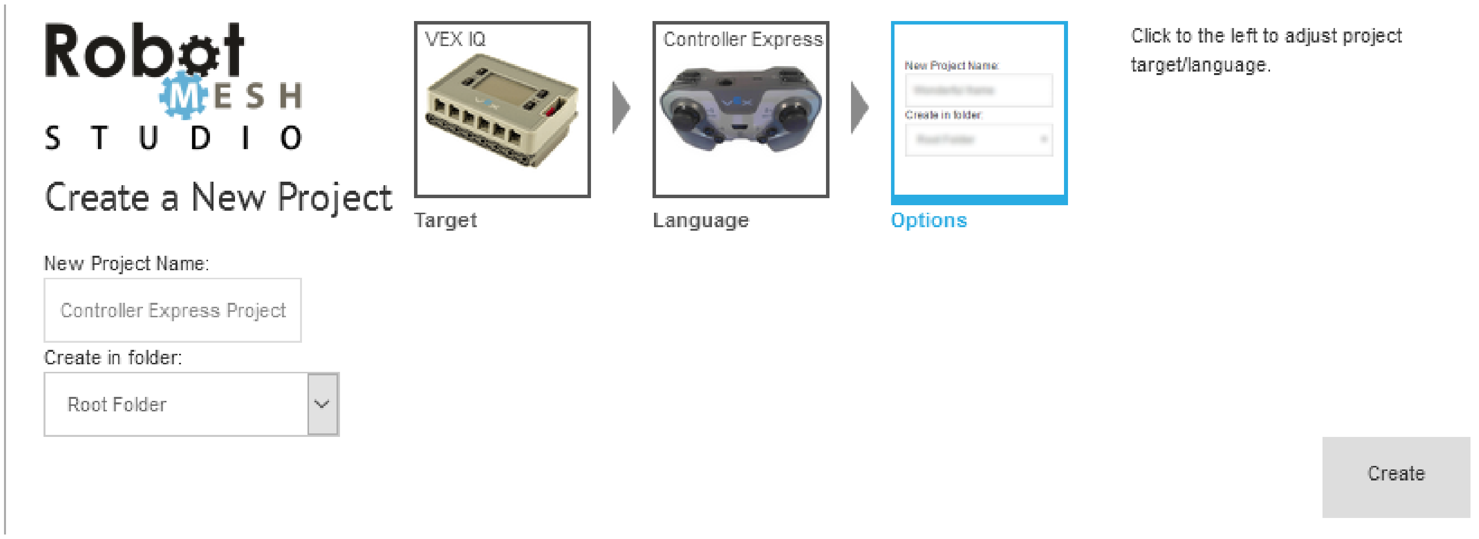


VEX IQ Standard Drive Base

Student Guide

After your robot is ready to go, go to Robot Mesh Studio at robotmesh.com, and create your project.VEX IQ robots are run by a computer called the VEX IQ Brain. The VEX IQ Controller sends inputs which the Brain then uses to execute actions. Using Controller Express, you can write a computer program in a few minutes to map inputs on the Controller to devices plugged into the VEX IQ Brain, based on your configuration.

Creating a New Project



Create a New Project

New Project Name: Controller Express Project

Create in folder: Root Folder

Before getting started make sure that you followed all the instructions in VEX IQ Controller Users Guide to configure your VEX IQ Brain, motors and sensors.

Prepare your robot:

1. Check to see that you have built the correct robot and that all motors and sensors are plugged into the correct ports
2. Check batteries
3. Install radios
4. Check to make sure the VEX IQ Brain is paired with Controller (if you are using the Controller)

Prepare RM Studio:

1. On the RM Studio project page click on **Create a New Project**
2. Click on **Target: VEX IQ** and **Language: Blockly, Flowol, Python or Controller Express**
3. Type your project name in the box. You might want to use the name of the activity as your project name
4. Click on **Create** and wait for the new project page to load
5. Plug your VEX IQ Brain into your computer using a USB A-to-micro cable (provided in the VEX kit)
6. Press the check mark button on the Brain to turn it on
7. Click on **Detect Sensors**. The panel on the right will now show two motors. If any other sensors were installed, those would show up, too
8. Check that the motors are plugged in and configured in the Device Monitor and Configuration panel
9. Change the name of motor_1 to motor_left, and motor_6 to motor_right

10. For "motor_right" click on the **Gear Icon**, select "Motor (reverse polarity)" in the Subtype box, and then click "**OK**."

Student Instructions

Now that you have created a programming project and configured your motors, you can use Controller Express to tell your robot what to do:

1. On the left side of the screen, click on **Add Action** next to **Axis A**. Then, click on the **Gear Icon** and click on **Linear Control**.
2. Next to **Motor**, select "motor_left." Don't click on the "X" - clicking here deletes your choices and you will need to start over.
3. Repeat the above steps for the right motor on channel D.

Downloading and Testing Your Program

1. Your robot should still be plugged in, but if it is not, plug it into your computer before continuing. Make sure the VEX IQ Brain is turned on!
2. Click **Run** to download your program to the robot. You should now see "Download Done" in a green bar at the bottom of the screen.
3. Unplug your robot from the computer, and turn on your controller.
4. Once you see the Radio/Tethered status indicator on the Brain showing a good link, try moving the joysticks on the Controller to see what happens. Your robot should now be working as you programmed it.

Activity Extensions

In Robot Mesh Studio, click on the **Generated Code** tab. What you see there is the programming code that the Controller Express creates automatically. It is in Python, a high-level programming language that is popular in both schools and industry, combining ease of learning with powerful features. We will be going into Python later in our curriculum, but it is good to remember that both Controller Express and Blockly create Python code, which is what is actually downloaded to the VEX IQ Brain.

Changing to Arcade Drive

The previous program uses what is called "tank drive" where each side of the drivetrain is controlled separately by each joystick. Some robot drivers prefer what is called "arcade drive" where one joystick controls both drive motors. To test arcade drive, do this:

1. In the Controller Express panel, click the "X" marks for each motor to clear the motor assignments.
2. Click on **Add Action** next to **Axis A**. Then, click on the "v" and click on **Arcade Drive**.
3. Download your program following the directions above.

Try driving your robot. Does the C/D joystick do anything? How does the robot respond to the A/B joystick?
Response:
What happens when you set the arcade settings on channel D instead of A? Can you figure out how to try this?
Response:

Post-Activity Checkup

1. What do the letters printed on the VEX IQ Controller mean? How do you use them in programming?
Response:
2. What is the difference between tank drive and arcade drive? Which do you think is easiest to drive?
Response:
3. What kind of programming code does Controller Express create?
Response:
Don't take this robot apart, we will use it in the next unit!

3. Programming Basic Movement Activity

Activity Overview

In this unit, you will use Python to program a robot that moves forward, using both basic motor commands and the drivetrain function. You will also learn the basics of building a Python program, and how to change your program.

Activity Preparation

This Unit uses the Standard Drive Base that you built in the last unit. If you do not have a robot, please follow build steps 1-19 in VEX IQ Build Instructions.
Before getting started make sure that you followed all the instructions in VEX IQ Controller Users Guide to configure your VEX IQ Brain, motors and sensors.
Prepare your robot:

1. Check to see that you have built the correct robot
2. Check batteries
3. Install radios
4. Check to make sure the VEX IQ Brain is paired with Controller (if you are using the Controller)

Prepare RM Studio:

1. On the project page click on **Create a New Project**
2. Click on **Target: VEX IQ** and **Language: Python**
3. Type your project name in the box. You might want to use the name of the activity as your project name
4. Click on **Create**
5. Plug your VEX IQ Brain into your computer using a USB A-to-micro cable (provided in the VEX kit)
6. Press the check mark ü button on the Brain to turn it on
7. Click on the **Refresh** button, and then on the **Connect** button in Robot Mesh Studio
8. Click on **Detect Sensors**. The panel on the right will now show two motors. If any other sensors were installed, those would show up, too
9. Check that the motors are plugged in and configured in the Interface Monitor

10. Change the name of motor_1 to motor_left, and motor_6 to motor_right

Student Instructions

Getting Started

In this lesson, we will use Python to program our robot autonomously. In the last unit we used the Controller to send commands to the robot, and in this activity you will create an autonomous program that the robot will follow without your help.

Three Important Ideas

Autonomous programs run without human intervention. You write and download code, and the robot does what you programmed it to do.

Driver control programs combine input from the VEX IQ Controller and a program that you wrote to control the robot. You will learn in future Units how to combine programming with Controller input to make your robot better at its tasks.

Algorithms are step-by-step instructions for solving a problem or completing a task. Computer programmers use programming languages to create algorithms that their computer or robot will execute. Whenever you program your robot, you are inventing and testing algorithms.

Introduction to Python

Up to this point, you have followed the same steps as you did in the Controller Express unit, but now we will start programming. When you create a Python VIQ project, you will see a few lines of code are already included. Do not delete or change the second or third lines (the "import" commands):

```
# VEX IQ Python-Project
import sys
import vexiq
```

"Sys" is a library of Python functions, and "vexiq" contains any system function that talks to the VEX IQ hardware. If you delete these, your code will not work. The first line, starting with "#," is a

comment. We will talk more about comments later.
In addition to "vexiq" and "sys," which Python added for you, when you configured your motors by clicking on "Detect Sensors" and then changing their names, Python added them to a section called the "#region config." This creates instances of the motor objects defined by vexiq.motor() and names them "motor_left" and "motor_right", as you requested. We will talk more about "instances" and "objects" in a later activity.

```
# VEX IQ Python-Project
import sys
import vexiq

#region config
motor_left  = vexiq.Motor(1)
motor_right = vexiq.Motor(6)
#endregion config
```

If you have other devices connected, you will see more definitions in the "config" region.
Your Python program will start at the top of your program and then proceed to run the steps in order. At the end of the program, the Brain will shut off motors and all connected sensors, and stop "listening" to the Controller. In our program, we turn the motors on, but since the program ends right after that, the Brain will turn off the motors immediately. Your robot will not move.
Let's make the robot move using the two drive motors. To do this, we will use the VEX IQ Motors function.
You will start with adding some code after the #endregion config statement. With Python,
Motors are controlled using the object "vexiq.motor." In the Device Monitor panel in RM Studio, we have already setup the motors such that motor_left is plugged into Port 1, and motor_right is plugged into port 6.
Add motor commands to your program using the motor object and its properties. For this simple project we will be using "motor.run, which you can learn more about at http://www.robotmesh.com/docs/vexiq-python/html/classvexiq_1_1_motor.html. The "run" method is only one option you have for motors in Python, and it is used like this:
motor_left.run(50) where "motor_left" is the motor we want to use, "run" is the method we want to execute, and "50" is the power we want to use, 50% of maximum.
Enter the following in your program, after the #endregion config line.

```
# main thread
motor_left.run(50)
```

```
motor_right.run(50)
motor_left.off()
motor_right.off()
```

You now have a program that will turn on both your motors. What do you think will happen when you run it on the robot? Let's download the code and see what happens.

Downloading and Testing Your Program

1. Plug your robot into your computer, and make sure the VEX IQ Brain is turned on.
2. Caution! Your program will run as soon as you download it, so make sure your robot is on the floor or someone is holding it. You do not want to drive it off the table!
3. Click **Run** to download your program to the robot. You should now see "Download Done" in a green bar at the bottom of the screen.
4. Unplug your robot, and set it on the floor.
5. On the Brain, use the up-arrow and down-arrow buttons to highlight the program you downloaded.
6. Making sure that you robot is in a safe area and no one has their hands or eyes near it (wear safety glasses), push the check-mark button on the Brain to start your program. Run it more than once if you want to.

What Went Wrong?

Nothing, really, but the robot probably did not do what you thought it would. Let's go through this one step at a time.
If you look at your program, what it did was:

1. Import system files
2. Create motor object instances
3. Turned each motor on
4. Turned each motor off

Notice that there is no gap between steps 3 and 4. You turn the motors on and then turn them off right away. An easy way to deal with this is to have the Brain pause while the motors run. Try adding a sys.sleep() function as shown below, then download and run the program on your robot. The number in the sys.sleep() function is the number of seconds the brain will pause, in this case three.

```
#VEX IQ Python-Project
import vexiq
import sys

#region config
motor_left = vexiq.Motor(1)
motor_right = vexiq.Motor(6)
#endregion config

# main thread
motor_left.run(50)
motor_right.run(50)
sys.sleep(3)
motor_left.off()
motor_right.off()
```

What happened this time? Did your robot do what you thought it would. Probably, you saw the robot turn clockwise (to the right, looking straight down) and not go straight. Which direction was the left motor turning? The right motor? Why do you think it turned instead of going straight?

	Pro Tip: VEX prints the direction of rotation on the end of VEX IQ motors. You can always look at the motor for a quick review of direction.

If you wanted your robot to go straight, you will need to do something different.

1. Both motors are turning clockwise, but one is mounted on the left and one on the right. To the robot chassis, it "looks" like they are going opposite directions, even though both motors turn clockwise. You can experiment with this by looking at your robot while turning the wheels by hand. If you turn both of them clockwise (looking at the motor) would your robot go straight?
2. What would make the robot go straight?
3. Hopefully, you figured out that you need one of the motors to turn in the opposite direction. Since the left motor is already moving the left side of the robot in the correct direction, let's try changing the direction of the right motor.
4. Open up your project in Robot Mesh Studio, and edit the motor.run() function, changing the "50" to "-50." Setting a motor to a reverse value, it will run in the opposite direction.
5. Download your program and test it. What happened this time? Did your robot go straight?

Your program should now look like this:

```
import vexiq
import sys

#region config
motor_left = vexiq.Motor(1)
motor_right = vexiq.Motor(6)
#endregion config

# main thread
# Turn both motors on
motor_left.run(50)
motor_right.run(-50)
sys.sleep(3)
motor_left.off() #turn off left motor
motor_right.off() #turn off right motor

</code>
```

You just learned an important lesson for most robots, which is that to get the wheels moving in the same direction, sometimes you have to run motors backwards.

Comments

It's a good idea to include descriptions of what is going on in your program, inside the code. In Python, you use a "#" symbol to mark a comment. Text in a comment turns green, and has no effect on the program. You can put comments on a line by themselves, or on the same line as program code, AFTER the code. You will see both kinds of comments in the code sample above.

A Better Way - the Drivetrain Feature

Python for VEX IQ includes a special feature called the drivetrain which gives you a simpler way to run your drive motors. Try this:

1. Start a new project, and call it "Drivetrain Test."
2. Plug your robot into your computer, click on "Connect" and "Detect Sensors."
3. Change the names of the motors on port 1 and port 6 to "motor_left" and "motor_right" respectively.
4. For "motor_right" click on the "v" and select "Motor (reverse polarity)"

5. On the far right, underneath the motor definitions, click on the gear next to **Configure drivetrain here**.
6. Click the **Enabled** button.
7. In the **Left Motor** box pick "motor_left" and in the **Right Motor** box, pick "motor_right."
8. Leave the **Wheel Travel** and **Track Width** alone for now.
9. Click on **OK**.
10. You will now notice this code in your program. This imports the drivetrain system file, and defines a drivetrain object called "dt" as a drivetrain:
    ```
    import drivetrain
    dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 176)
    ```
11. Add this to your code
    ```
    dt.drive_until(50,1000)</code>
    ```
12. This means, "use the dt object, and the drive_until method to set the robot's speed to 50 (percent) and go for 1000mm (about 39 inches). There are other methods for the drivetrain object, which you can explore in the Drivetrain Functions reference guide.

Your code will now look something like this:

```
# VEX IQ Python-Project
import sys
import vexiq

#region config
motor_left = vexiq.Motor(1)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 176)
#endregion config

dt.drive_until(50,1000)
```

What do you think your robot will do now?

Activity Extension Ideas

Click **Save** before doing these extension activities
1. Change the power setting to 100 in your drivetrain command and try your program again. What happens?

2. Change the distance to 100. Then try 10000. What happens each time?

3. In your first program, try setting sleep to 20 seconds. What happens?

Post-Activity Checkup

In this section we learned:
1. How do you put programs together in Python?

2. Is there an easier way to make the robot move than sending individual commands to the motors attached to the VEX IQ Brain? What is it?

3. What does the Sleep function do?

4. In the drivetrain function we used, there are two numbers. What do those numbers do?

4. Programming Drivetrains Activity

Activity Overview

In our first lesson, you learned how to turn on motors two different ways. Now, we are going to learn how to make motors go for a certain distance and introduce turning. When you are done, you will have a good grasp of the basics of both Python programming and how to make your robot move.

Activity Preparation

This Unit uses the Standard Drive Base that you used in the last activity. If you do not have a robot, please follow build steps 1-19 in VEX IQ Build Instructions.
Before getting started make sure that you followed all the instructions in VEX IQ Controller Users Guide to configure your VEX IQ Brain, motors and sensors.
Prepare your robot:

1. Check to see that you have built the correct robot and that all motors and sensors are plugged into the correct ports
2. Check batteries
3. Install radios
4. Check to make sure the VEX IQ Brain is paired with Controller (if you are using the Controller)

Prepare RM Studio:

1. On the RM Studio project page click on **Create a New Project**
2. Click on **Target: VEX IQ** and **Language: Blockly, Flowol, Python or Controller Express**
3. Type your project name in the box. You might want to use the name of the activity as your project name
4. Click on **Create** and wait for the new project page to load
5. Plug your VEX IQ Brain into your computer using a USB A-to-micro cable (provided in the VEX kit)
6. Press the check mark button on the Brain to turn it on
7. Click on **Detect Sensors**. The panel on the right will now show two motors. If any other sensors were installed, those would show up, too
8. Check that the motors are plugged in and configured in the Device Monitor and Configuration panel

9. Change the name of motor_1 to motor_left, and motor_6 to motor_right
10. For "motor_right" click on the **Gear Icon**, select "Motor (reverse polarity)" in the Subtype box, and then click "**OK**."

Student Instructions

Getting Started

In this lesson, we will continue to use Robot Mesh Python to program our robot autonomously. In the last unit we learned how to make your robot move in a straight line, and in this we learn more options.

Set up the Drivetrain function

1. For "motor_right" click on the "v" and select "Motor (reverse polarity)"
2. On the far right, underneath the motor definitions, click on the gear button next to **Configure drivetrain here**.
3. Click the **Enabled** button.
4. In the **Left Motor** box pick "motor_left" and in the **Right Motor** box, pick "motor_right."
5. Click on **OK**.

Motor Reverse Polarity

As we learned in the first Unit, to make your robot go straight, you usually have to have your drivetrain motors turn in opposite directions. You can do this either by sending a negative value to the "backwards" side, or you can check the "reverse polarity" option instead. To make the Drivetrain function work, you will have to click the reverse polarity option in motor set up.

Moving Autonomously

We are now going to use the Drivetrain function to show how you can advance for a defined distance or amount of time, turn, and then continue driving.

1. In your project, create a drivetrain function using 50% and set the distance to 1000 mm. This tells the robot to go forward for 1 meter (about 39 inches).

Your program should now look like this:

```
# VEX IQ Python-Project
import sys
import vexiq

#region config
motor_left = vexiq.Motor(1)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 176)
#endregion config

#Main
dt.drive_until(50,1000)
```

In the last lesson, we used the Sleep function to give the robot time to finish its movement, but here we are using a Drive Power Until function, which behaves differently. In this program, we have told the robot to drive forward and not stop until it has traveled 1000 mm.

Hitting the Wall - Motor Stalling

What happens if your robot hits a wall or other stationary object before reaching the desired distance? If your robot hits an object that won't move, like a wall, it will stop moving, but the motors will "stall," meaning that you are applying power to the motors but they cannot turn. The Drive Until function include an automatic provision that if the motors stall the motors will switch off automatically. It's good to remember, though, that you might be driving on a slippery surface, and you could run your robot into a wall and the wheels keep spinning even though the robot is not moving. There is no simple way for the Brain to know that you are spinning your wheels, so the motors will keep turning until they reach their goal, or, in this case, 1000 mm.

You now have a program that will turn on both your motors and go forward for 1000 millimeters. What do you think will happen when you download your project and run it on the robot? Let's download the code and see what happens.

Downloading and Testing Your Program

1. Plug your robot into your computer, and make sure the VEX IQ Brain is turned on.
2. Caution! Your program will run as soon as you download it, so make sure your robot is on the floor or someone is holding it. You do not want to drive it off the table!
3. Click **Run** to download your program to the robot. You should now see "Download Done" in a green bar at the bottom of the screen.

4. Unplug your robot, and set it on the floor.
5. On the Brain, use the up-arrow and down-arrow buttons to highlight the program you downloaded.
6. Making sure that you robot is in a safe area and no one has their hands or eyes near it (wear safety glasses), push the check-mark button on the Brain to start your program. Run it more than once if you want to.

What Happened?

If you robot is built correctly and you entered the program above, you should have seen your robot drive forward for 1000 mm and then stop.
What did your robot do? Did it spin in circles, or even move slowly in an arc to the right or left? Check to make sure you selected reverse motor polarity for motor_right, and that your Brain is plugged in correctly.
You have now learned how to move your robot in a straight line, now we will add turning to the mix.

Going Left

We tried the "drive until" method on the drivetrain object to drive straight. To turn left, we use the "turn_until()" method. Click here to learn about all drivetrain functions and methods.

1. Add the **Turn Until Angle** function to your program using 50 for the power and 90 degrees for the angle.
2. Add another **Drive Until Distance** function below this. Change the values in the function to go 1000mm at 50 power.

Your code should now look something like this:

```
# VEX IQ Python-Project
import sys
import vexiq

#region config
motor_left = vexiq.Motor(1)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 176)
#endregion config
```

```
# Main
dt.drive_until(50,1000)
dt.turn_until(25,90)
dt.drive_until(50,1000)
```

What do you think this is telling your robot to do? Just walk through the steps and can figure out what the program is doing. Download this and test it on your robot, using the steps above.
What did your robot do? Did it drive forward, turn, then drive forward again? If so, your code worked!
Ready to make your robot come back to its starting point?

Driving in a Square

In the last program your robot turned only once, but in this one, your robot will turn four times and come home. Take the program you have been working on, and add more drivetrain turn and move functions. Change the power to 50, and distance to 500 for all the "drive until" functions, and change the power to 50 in the "turn until" function. Your program should look something like this when you are done:

```
# VEX IQ Python-Project
import sys
import vexiq

#region config
motor_left = vexiq.Motor(1)
touch_led_2 = vexiq.TouchLed(2)
color_3 = vexiq.ColorSensor(3) # hue
distance_4 = vexiq.DistanceSensor(4, vexiq.UNIT_CM)
gyro_5 = vexiq.Gyro(5)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
bumper_8 = vexiq.Bumper(8)
bumper_9 = vexiq.Bumper(9)
import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 230)
#endregion config

# Main
dt.drive_until(50,500)
dt.turn_until(25,90)
dt.drive_until(50,500)
dt.turn_until(25,90)
dt.drive_until(50,500)
dt.turn_until(25,90)
dt.drive_until(50,500)
dt.turn_until(25,90)
```

Walk through this program and see if you can figure out what it is doing, then download it and test it. What happened?

You might have a robot that did not make a perfect circle, but that is OK. Not all robots are the same, and all floors are not either. If your robot is turning too far, or not enough, you want to go back to the Drivetrain configuration panel, and modify the Track Width setting. If your robot is turning too much, make the number smaller, and if it is not turning enough, make it bigger. We suggest only changing the number by 5 mm at a time, and then downloading and checking your code. On our test robot on tile floors, we ended up changing the track width to 230 mm to get a perfect square.

Activity Extension Ideas

Click **Save** before doing these extension activities
1. What if you wanted to make your robot move left, then right, then left, then right again? Change your code and try it. What happened?

2. What if the angle is not 90 degrees? Try some other settings and test your new code.

3. Make some changes to the power settings and see what happens. What happens if set the power really low, like 20? What happens when you change the distance to a negative number (-500 instead of 500). Give it a try and let us know what you found.

Post-Activity Check-Up

1. In the second exercise, did your "Driving in a Square" robot end up exactly where you thought it would? If not, why do you think it did not return exactly where it started? Can you think of a way to change the values in the function or in the drivetrain settings to make it more perfectly a square?

2. We have learned two ways to tell a robot to move and then stop. What are the two functions that we used to tell the robot to go then stop? In which functions are these used?

3. What do you think the **Turn Until Angle** function does? What do the drive motors do in this function?

4. What do you think the **Drive Until Distance** function does? What do the drive motors do in this function?

5. Programming Drivetrains Continued Activity

Lesson Overview

You are well on your way to being an ace robot programmer, but there are a few more things to learn about drivetrains first. In this unit you will learn about:

1. Two different kinds of drivetrain functions
2. Driving without stopping
3. Turning without stopping
4. Writing a program to do the same thing multiple times

Activity Preparation

This Unit uses the Standard Drive Base that you used in the last unit. If you do not have a robot, please follow build steps 1-19 in VEX IQ Build Instructions.
Before getting started make sure that you followed all the instructions in VEX IQ Controller Users Guide to configure your VEX IQ Brain, motors and sensors.
Prepare your robot:

1. Check to see that you have built the correct robot and that all motors and sensors are plugged into the correct ports
2. Check batteries
3. Install radios
4. Check to make sure the VEX IQ Brain is paired with Controller (if you are using the Controller)

Prepare RM Studio:

1. On the RM Studio project page click on **Create a New Project**
2. Click on **Target: VEX IQ** and **Language: Blockly, Flowol, Python or Controller Express**
3. Type your project name in the box. You might want to use the name of the activity as your project name
4. Click on **Create** and wait for the new project page to load
5. Plug your VEX IQ Brain into your computer using a USB A-to-micro cable (provided in the VEX kit)
6. Press the check mark button on the Brain to turn it on

7. Click on **Detect Sensors**. The panel on the right will now show two motors. If any other sensors were installed, those would show up, too
8. Check that the motors are plugged in and configured in the Device Monitor and Configuration panel
9. Change the name of motor_1 to motor_left, and motor_6 to motor_right
10. For "motor_right" click on the **Gear Icon**, select "Motor (reverse polarity)" in the Subtype box, and then click "**OK**."

Student Instructions

In this lesson, we will continue to use Python to program our robot autonomously. In the last unit we learned how to make your robot make turns, and in this we learn more drivetrain options.

Set up the Drivetrain function

1. For "motor_right" click on the "v" and select "Motor (reverse polarity)"
2. On the far right, underneath the motor definitions, click on the gear button next to **Configure drivetrain here**.
3. Click the **Enabled** button.
4. In the **Left Motor** box pick "motor_left" and in the **Right Motor** box, pick "motor_right."
5. Click on **OK**.

Drivetrain Functions with Limits

You should remember the two drivetrain methods (or "functions") that we used in the last unit:

```
dt.drive_until(50,500)
dt.turn_until(25,90)
```

These two functions work the same way, they will run until the condition is met. For the **Drive Until Distance** method, the robot will travel the indicated number of millimeters, and the **Turn Until Angle** method will keep the robot turning until the angle target is met.

Drivetrain Functions without Limits

These options give you a little more flexibility, but they also require a little more programming. All of these functions will start an activity, but will stop executing when the next line of code starts. You will need to write the code to make these work for you.

dt.drive(p) where p is power

dt.drive(p,d) where p is power and d is distance

dt.turn(p,a) where p is power and a is angle (negative to turn right, positive to turn left)

This will require the use of some additional drivetrain functions:

dt.reached_target()

dt.stall_timeout(t) where t is the stall detection timeout (in seconds)

dt.stalled() this is either True or False

For now, the important additions are the Wait functions, implemented as **sys.wait_for**. Here is how to use them:

dt.drive(p,d) where p is power and d is distance

sys.wait_for(dt.reached_target)

and

dt.turn(p,a) where p is power and a is angle (negative to turn right, positive to turn left)

sys.wait_for(dt.reached_target)

You can replace "dt.reached_target" with "dt.reached_target_or_stalled" if you want to check that your robot has EITHER stalled OR reached the target distance:

sys.wait_for(dt.reached_target_or_stalled)

In later activities, you will have a chance to try out some of the other functions, but for now just remember that some drivetrain functions require logic to make them wait, and some other functions include their own "do this until this is reached" functionality.

Moving and Turning in a New Way

We are now going to use the Drivetrain functions to show how you can advance and turn using something other than the distance and time properties of functions.

1. On the Robot Mesh Studio programming screen, add a drivetrain object, with the ".drive" method, and set it for 100 power.
2. Add a 3-second sys.wait function.
3. Add a drivetrain object, with the ".turn" method, and set it for .

Your program should look like this:

```
# VEX IQ Python-Project
import sys
import vexiq

#region config
motor_1 = vexiq.Motor(1)
motor_6 = vexiq.Motor(6, True) # Reverse Polarity
import drivetrain
dt = drivetrain.Drivetrain(motor_1, motor_6, 200, 176)
#endregion config

dt.drive(100,300) #drive straight at 50% for 1000mm
sys.sleep(3) #sleep for 3 seconds
dt.turn(50) #turn right 90 degrees
sys.sleep(3) #sleep for 3 seconds
```

As you can see, we are using the Sleep function to give the robot time to follow our instructions, instead of the Distance property in the **Drive Until Distance** and **Turn Until Angle** methods. It can be useful if you are not sure how far you want to travel, but want to travel for a fixed period of time.

Downloading and Testing Your Program

1. Plug your robot into your computer, and make sure the VEX IQ Brain is turned on.
2. Caution! Your program will run as soon as you download it, so make sure your robot is on the floor or someone is holding it. You do not want to drive it off the table!
3. Click **Run** to download your program to the robot. You should now see "Download Done" in a green bar at the bottom of the screen.
4. Unplug your robot, and set it on the floor.
5. On the Brain, use the up-arrow and down-arrow buttons to highlight the program you downloaded.
6. Making sure that you robot is in a safe area and no one has their hands or eyes near it (wear safety glasses), push the check-mark button on the Brain to start your program. Run it more than once if you want to.

What Happened?

If you robot is built correctly and you entered the program above, you should have seen your robot drive forward for 3 seconds, then turn left and stop. Is that what you saw?
You can change the time or power settings and see how that affects the program.

Stalling Your Drive Motors

Take your robot that you just programmed, place it a short distance from a wall (careful - the robot might scratch the wall!), and see what happens. The robot was stuck against the wall, but still trying to run, wasn't it? What you are seeing is the motors stall because they were told to go, but could not. Stalling is when a motor is trying to turn, but it cannot.
The motors in a VEX IQ robot are direct current (DC) battery-powered motors, and all DC motors use the most power and get the hottest when told to run at full power with the motor stalled out and not moving. This is bad for several reasons, the worst of which is that constantly stalling DC motors will eventually damage them. In VEX IQ we see this most when a motor is being programmed to move and cannot, and when a motor is asked to do more than it can.
To help you manage motor stalling and to protect your motors from damage, Robot Mesh has added some stall functions to Python. Using Powertrain options, you have these functions available:

- Drivetrain Set Stall Timeout - This will automatically shut off your drivetrain after the specified amount of time if it is stalled.
- Wait Until Drivetrain Stalled - This is similar to Wait Until Drivetrain Reached Target you used before, except that the "wait" will continue until the drivetrain stalls.
- Wait Until Drivetrain Reached Target or Stalled - This will keep the drivetrain running until it either stalls or reaches its target.

Let's see how these work on your robot.

Motor Stalling Activity

We are going to use the "drive_until" method along with the stall timeout for this program. Change the distance to 1000 mm. We are only changing the dt.drive statement and dt.turn statements.
Your program should look like this:

```
# VEX IQ Python-Project
import sys
import vexiq
```

```
#region config
motor_1 = vexiq.Motor(1)
motor_6 = vexiq.Motor(6, True) # Reverse Polarity
import drivetrain
dt = drivetrain.Drivetrain(motor_1, motor_6, 200, 176)
#endregion config

dt.drive(50,1000) #drive straight at 50% for 1000mm
sys.wait_for(dt.reached_target_or_stalled)
dt.turn(50,-90) #turn right 90 degrees
sys.wait_for(dt.reached_target_or_stalled)
```

Download your program and test it by placing the front of your robot against a wall or even your foot. Turn it on and see what it does. If your robot is like most, the wheels will keep spinning until they think they have traveled 1000 millimeters. If you want to test the stall feature, press down firmly on the top of your robot, and then turn it on. The stall feature should turn it off in a couple of seconds, or even faster. Don't worry, you will not hurt it - that's what the stall detector is for.
All of the stall functions and properties are there to let you know your motors are in danger, and that your motor is not moving. When would you use the stall properties? First, in a drivetrain to let you know that you have run into something, or, second, to let you know that an arm or other mechanism is stuck or has run into a barrier. It is always a good idea to include stall checking in your code.

Driving in a Square

In the last unit, we made our robot turn in a square pattern, driving, turning, and repeating this three more times. To remind you, your program should have looked similar to this:

```
# VEX IQ Python-Project
import sys
import vexiq

#region config
motor_left = vexiq.Motor(1)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 230)
#endregion config
```

```
# Main
dt.drive_until(50,500)
dt.turn_until(25,90)
dt.drive_until(50,500)
dt.turn_until(25,90)
dt.drive_until(50,500)
dt.turn_until(25,90)
dt.drive_until(50,500)
dt.turn_until(25,90)
```

Repeating the Same Action Using a Loop

You can see that the program above consists of a set of Drive and Turn functions one after another. What if your robot had to do this twice, how would you add that to the program? Would you add eight more functions?

There is a much better way, and it uses the Repeat function in Python. Let's build a program that uses the Repeat function to achieve what we did in the last activity, with a lot less code. Start by creating a new project and then build your new program.

For this project you will use the **For** statement. According to the Python Wiki:

"For" loops are traditionally used when you have a block of code which you want to repeat a fixed number of times. The Python for statement iterates over the members of a sequence in order, executing the block each time.

The structure of a for loop in Python is:

For <variable name> in <range>, do these things once for each value in the range.

For loops can do other things in Python (like working with lists of text), but that is not what we need here. Let's keep it simple.

Let's describe our program in English:

1. create the motors
2. setup the drivetrain
3. go forward at 50% for 500mm
4. turn left 90 degrees
5. repeat the forward and turn three more times

We will use the variable name "count" for the **For** statement. Here is what it will look like in our program. Create a Python project, configure left and right motors, create a drivetrain, and enter this code into your project:

Your code should now look like this:

```
# VEX IQ Python-Project
import sys
import vexiq

#region config
motor_left = vexiq.Motor(1)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 176)
#endregion config

# main
for count in range(4):
   dt.drive_until(50, 500) #note the indented code
   dt.turn_until(50, 90) #note the indented code
dt.off()
```

Note that the code in the **for** loop is always indented. Python assumes that the indented code makes up the repeated functions, and when the **for** loop is done, Python will go on to the next non-indented code.

The Range Statement

As you learn more about Python, you will find that **range** is used in other places than in **for** loops. For now, know that range paired with for gives you a powerful way to go through a set of steps a fixed number of times -- one for each item in the range.

On more thing about ranges. In our Python program, we set the range to "4." We could have also written this as **range (0,4)** which means exactly the same thing: go through this loop four times, starting with 0 (zero) and going to 3. Unless you tell it otherwise, Python assumes **range(4)** means "start with zero, and go up by one integer each pass through the loop, ending when you get to 4. When you get to 4, skip the loop and go to the next program instruction." KEY FACT: the biggest number in a range marks when processing jumps out of the loop and goes on to the next thing. If you want something to happen four times, do NOT use range(1,4) because then the loop will process 1, 2, and 3, and then skip on 4. The biggest integer number in the range should be one higher than the last number you want to use in the loop.

Download and test your program. What did the robot do? Was it the same result as the program above that had 8 functions?
Response:
Change your **for** loop to **range(6)** and run your program again. What did your robot do? We will use this for loop function many times in programming VEX IQ robots. Watch for some new loop functions in future Units.

Activity Extension Ideas

Click **Save** before doing these extension activities
1. Try to program this unit exercise using Motor command functions instead of the Drivetrain functions. Was it easier or harder using Motor functions? How did your program work?

2. Try replacing the function "drivetrain reached target or motor stalled" with the "drivetrain reached target" function. Were able to figure out how use click-drag-and-drop to make this change?

Post-Activity Check-Up

1. List three ways that using a loop is better than just pasting the same functions in over and over again.

2. In your own words, tell us what the **for** loop does, and describe one way to use it.

6. Bumping Along with Bumper Switches Activity

Lesson Overview

In this unit you will learn about:

1. Sensors in general
2. How to use a sensor to control a robot's actions
3. Using "If" statements
4. Using the Timer

Activity Preparation

This is your first project using the Autopilot Robot from Chapter 4 of the **VEX IQ Build Instructions**, so you will need to build this robot before you start programming. See build steps 102-117 for instructions (page 32).
Before getting started make sure that you followed all the instructions in VEX IQ Controller Users Guide to configure your VEX IQ Brain, motors and sensors.
Prepare your robot:

1. Check to see that you have built the correct robot and that all motors and sensors are plugged into the correct ports
2. Check batteries
3. Install radios
4. Check to make sure the VEX IQ Brain is paired with Controller (if you are using the Controller)

Prepare RM Studio:

1. On the RM Studio project page click on **Create a New Project**
2. Click on **Target: VEX IQ** and **Language: Blockly, Flowol, Python or Controller Express**
3. Type your project name in the box. You might want to use the name of the activity as your project name
4. Click on **Create** and wait for the new project page to load
5. Plug your VEX IQ Brain into your computer using a USB A-to-micro cable (provided in the VEX kit)
6. Press the check mark button on the Brain to turn it on

7. Click on **Detect Sensors**. The panel on the right will now show two motors. If any other sensors were installed, those would show up, too
8. Check that the motors are plugged in and configured in the Device Monitor and Configuration panel
9. Change the name of motor_1 to motor_left, and motor_6 to motor_right
10. For "motor_right" click on the **Gear Icon**, select "Motor (reverse polarity)" in the Subtype box, and then click "**OK**."

Student Instructions

Getting Started

In this lesson, we will continue to use Python to program our robot autonomously. In the last unit we learned how to make your robot make turns, and in this we learn more drivetrain options and how to use sensors and programming logic.

Set up the Drivetrain function

1. For "motor_right" click on the "v" and select "Motor (reverse polarity)"
2. On the far right, underneath the motor definitions, click on the gear button next to **Configure drivetrain here**.
3. Click the **Enabled** button.
4. In the **Left Motor** box pick "motor_left" and in the **Right Motor** box, pick "motor_right."
5. Click on **OK**.

Fun With Loops

We are going to use the **while** statement in this unit. It is similar to the **for** loop statement we used in the last unit. Briefly, **while** lets you write some code that will run as long as something is true (repeat while) or runs as long as something is not yet true. Here is the basic format of the command:

```
while not dt.drive.stalled:
   dt.drive(50)
```

This program would keep repeating the dt.drive(50) command until the drivetrain is stalled. As long as the drivetrain is not stalled (while not dt.drive.stalled) it will repeat. In other words, the robot will keep running until it hits something.

The VEX Bumper Switch

We will also use the VEX IQ Bumper Switch, the first of the VEX IQ Sensors. Your robot has two of these, one plugged into port 8 and one into port 9. To use the bumper switch, you will use the **bumper.is_pressed** statement.

VEX Bumper Switch

Let's build a program:

1. Type in **while not bumper_8.is_pressed:** with the colon at the end.
2. Press the tab key once, and type **dt.drive(-50)**
3. Type **dt.off()**

The bump switches are on the back of the robot, so we will run in reverse instead of forward, which is why the drivetrain command is "-50" instead of "50."
Your program should now look like this:

```
# VEX IQ Python-Project
import sys
import vexiq

#region config
motor_left = vexiq.Motor(1)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
bumper_8 = vexiq.Bumper(8)
bumper_9 = vexiq.Bumper(9)
import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 176)
#endregion config
```

```
# main thread
while not bumper_8.is_pressed():
   dt.drive(-(50))
dt.off()
```

What do you think will happen? Let's test your program and find out:

Downloading and Testing Your Program

1. Plug your robot into your computer, and make sure the VEX IQ Brain is turned on.
2. Caution! Your program will run as soon as you download it, so make sure your robot is on the floor or someone is holding it. You do not want to drive it off the table!
3. Click **Run** to download your program to the robot. You should now see "Download Done" in a green bar at the bottom of the screen.
4. Unplug your robot, and set it on the floor.
5. On the Brain, use the up-arrow and down-arrow buttons to highlight the program you downloaded.
6. Making sure that you robot is in a safe area and no one has their hands or eyes near it (wear safety glasses), push the check-mark button on the Brain to start your program. Run it more than once if you want to.

What Happened?

If you robot is built correctly and you entered the program above, you should have seen your robot drive (backwards) in a straight line until the bumper switch on port 8 touched something. What happens after the bump switch is activated?
You should have seen your robot stop moving when it ran into an obstacle. Why did this happen?
Here is what your robot program did:

- Started
- Read a while loop that told it to do everything in this loop until **bumper_8** is pressed.
- When **bumper_8** is pressed, the program goes on to the next statement after the **While** loop.
- The program then turns off the drivetrain and stops running.

That is kind of cool, you programmed your robot to stop when the bumper switch is pressed. How about if your robot played a little song before it stopped? We can use a function called vexiq.sound_wave to play a sound file. There are 16 pre-configured sound files, numbered 0,1,2 to 15.

1. Type a **vexiq.sound_wave()** function into your project as the last line.

Your program should now look like this:

```
# VEX IQ Python-Project
import sys
import vexiq

#region config
motor_left = vexiq.Motor(1)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
bumper_8 = vexiq.Bumper(8)
bumper_9 = vexiq.Bumper(9)
import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 176)
#endregion config

# main thread
while not bumper_8.is_pressed():
   dt.drive(-(50))
dt.off()
vexiq.sound_wave(0)
```

What happened? Did anything change in the way your program ran? Go ahead and play some of the other music choices if you have time.

Using the "If" Function to Make Decisions

The **if** function lets you check something, then have your robot do something based on that check. For example, you could program your robot to go forward until a bumper switch is pressed, then stop and play a wave sound file.
The **if** function is made up of two parts: the condition that is checked, and what happens as a result of that check. For example:
If the bumper switch is pressed
<do this code>
If it is not pressed exit the if and do the next step

Evaluating True or False

In general, what an if statement does is say that if this statement is true, then do this. If it's not true, go on to the next. The question, though, is what is "true" and what is "false." Here are some examples for you:

- Bumper8.is_pressed is "true" if the button is depressed, and false if it isn't.
- If the If expression is "bumper8.is_pressed or bumper9.is_pressed" the truth of the expression is "true" if either one is pressed. In other words, with an "or" only one of the cases has to be true to make the whole expression true.
- If the If expression is "bumper8.is_pressed and bumper9.is_pressed" the truth of the expression is true only if BOTH of the cases is true. In other words if neither is pressed or only one is pressed, the value of the expression is false.

Make the Robot Smarter

You can now make a robot stop when it runs into something, but wouldn't you rather have it do something more fun? When the robot bumps into an obstacle, let's make it back up and drive off in a new direction, and let's use that other bumper switch, too. You can either use your existing project and change the code, or you can start a new project. If you choose to start a new one, don't forget to detect sensors, rename the motors and configure the drivetrain following the directions above.

1. Start with a **while True:**
2. Start the robot moving backwards using **dt.drive(-50)**, indented below the while
3. Add an "if" statement along with **bumper_8.is_pressed() or bumper_9.is_pressed():**
4. Indented in the "if" function, add a drive statement and a turn statement
5. For fun, add a vexiq.sound_wave(8) function
6. No need to turn the drivetrain off, this robot won't stop until you turn it off

Your program should now look like this:

```
# VEX IQ Python-Project
import sys
import vexiq

#region config
motor_left = vexiq.Motor(1)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
bumper_8 = vexiq.Bumper(8)
bumper_9 = vexiq.Bumper(9)
import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 176)
```

```
#endregion config

# main thread
while True:
   dt.drive(-(50))
   if bumper_8.is_pressed() or bumper_9.is_pressed():
      dt.drive_until(50, 100)
      dt.turn_until(50, 90)
      vexiq.sound_wave(8)
```

Download it and see how it works. Did you notice anything funny about this program? One strange thing is that the program never ends, as long as there is power in the battery, the robot will keep moving. Wouldn't we like the robot to stop when we want it to?
In a computer program, there are just a few ways a program will come to a halt. First, you can tell it to do something a certain number of times. Second, it can be doing something over and over until it runs out of work to do. Third, you can use a timer and have it run for a certain amount of time. Let's try the third method.

Using a Timer Block to Halt a Program

Python has a timer, so we can tell the program to run for a certain number of seconds. Let's write a program to run our bumper code for 10 seconds. To do this, we will need to keep track of how long the program has been running, and then stop the code when that number equals our goal, which is 10 seconds.
Change your program to look like this. The only difference is that we replaced the **Repeat Forever** loop with a **Repeat Until** loop and plugged in the Timer function. Run your new program.

```
# VEX IQ Python-Project
import sys
import vexiq
import timer

#region config
motor_left = vexiq.Motor(1)
touch_led_2 = vexiq.TouchLed(2)
color_3 = vexiq.ColorSensor(3) # hue
distance_4 = vexiq.DistanceSensor(4, vexiq.UNIT_CM)
gyro_5 = vexiq.Gyro(5)
```

```
motor_right = vexiq.Motor(6, True) # Reverse Polarity
bumper_8 = vexiq.Bumper(8)
bumper_9 = vexiq.Bumper(9)
import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 176)
#endregion config

timer_ = timer.Timer()
# main thread
timer_.start()
while not timer_.elapsed_time() > 10:
   dt.drive(-(50))
   if bumper_8.is_pressed() or bumper_9.is_pressed():
      dt.drive_until(50, 100)
      dt.turn_until(50, 90)
      vexiq.sound_wave(8)
```

What did you see? Did you hear the music play every time your robot ran into something?

In the next activity we will learn more about loops, and about the advanced options in the If Then logical test.

Activity Extension Ideas

Click **Save** before doing these extension activities
In the project window, click on **Generated Code**. When you compile and download your Python program, Robot Mesh Studio converts your Python program to Python. You can see the Python code in the Generated Code section of the screen. Can you find your Python commands in the Python code?
1. We suggested writing the code so that the robot backs up 200mm, then turns left, then starts moving again. How would you change the code to try different ideas? Go ahead and make your changes, then test them and see what happens.

2. In the **Play Wave** block, there are lots of options. Try changing them and seeing what else you can play. Try some of the other VEX IQ Sound blocks. Can you get all of them to work at the end of this program?

3. You probably noticed that we go "reverse" to drive forward, and "forward" to go backwards. This is

because the bumper switches are mounted on the back of the robot. Using what we learned about motors in the early units, how would you change the program or motor setup so that "forward" goes forward and "reverse" goes backwards? Apply your changes to the program and test it. Did it work?

Post-Activity Check-Up

1. Where would you use a timer loop?

2. In the "if" function, we used a logical comparison to check if the bump switches had been pressed by the robot running into something. In your own words, what does this statement do if neither switch is pressed, only one switch is pressed, and if both are pressed.

3. How would this be different if the statement said "Bumper_8 is pressed AND Bumper_9 is pressed?"

4. Why did we set the drivetrain to "reverse" to go backwards?

5. How would you go about changing the volume level of the song that plays? (Hint: click on the "help" tab and type "sound volume" into the search box.)

6. We used sound to tell us that the robot had stopped. What else could you use sound for on a robot?

7. Bumping Along Loops and Logic Activity

Activity Overview

In this unit you will learn about:

1. How to use **If** statements to make your program even smarter
2. Advanced loops

Activity Preparation

This project uses the Autopilot Robot from Chapter 4 of the **VEX IQ Build Instructions**. You will need to build this robot before you start programming. See build steps 102-117 for instructions.
Before getting started make sure that you followed all the instructions in VEX IQ Controller Users Guide to configure your VEX IQ Brain, motors and sensors.
Prepare your robot:

1. Check to see that you have built the correct robot and that all motors and sensors are plugged into the correct ports
2. Check batteries
3. Install radios
4. Check to make sure the VEX IQ Brain is paired with Controller (if you are using the Controller)

Prepare RM Studio:

1. On the RM Studio project page click on **Create a New Project**
2. Click on **Target: VEX IQ** and **Language: Blockly, Flowol, Python or Controller Express**
3. Type your project name in the box. You might want to use the name of the activity as your project name
4. Click on **Create** and wait for the new project page to load
5. Plug your VEX IQ Brain into your computer using a USB A-to-micro cable (provided in the VEX kit)
6. Press the check mark button on the Brain to turn it on
7. Click on **Detect Sensors**. The panel on the right will now show two motors. If any other sensors were installed, those would show up, too
8. Check that the motors are plugged in and configured in the Device Monitor and Configuration

panel

9. Change the name of motor_1 to motor_left, and motor_6 to motor_right
10. For "motor_right" click on the **Gear Icon**, select "Motor (reverse polarity)" in the Subtype box, and then click "**OK**."

Student Instructions

Getting Started

In this lesson, we will continue to use Robot Mesh Blockly to program our robot autonomously. We will learn more about If logic and how you can use loops.

Set up the Drivetrain function

1. For "motor_right" click on the "v" and select "Motor (reverse polarity)"
2. On the far right, underneath the motor definitions, click on the gear button next to **Configure drivetrain here**.
3. Click the **Enabled** button.
4. In the **Left Motor** box pick "motor_left" and in the **Right Motor** box, pick "motor_right."
5. Click on **OK**.

"Else If (elif)" Makes If Statements Smarter

As you learned in the last unit, the **If** statement lets you check something, then have your robot do something based on that check. For example, you could program your robot to go forward 500 mm then stop and play a sound file.
In the last unit, you learned that the **If** statement works like this:

- Check the logic in the **If** statement. In our case for the last unit, we have the robot check to see if either bumper switch is pressed
- If one or both of the switches are pressed, run the program inside the **If** logic
- If neither is pressed, leave the **If** statement.

Pretty simple, right? The robot says, "at this point in the program, if this is true, do this stuff, otherwise just skip it." But what if you want the robot to do this instead: at this point in the program, check to see if bump switch 8 is pressed, and do this stuff if it is true, otherwise check to see if bump switch 9 is pressed and then do this other stuff, but if neither is pressed, it should just keep driving.

Here is a breakdown of what will happen with this **If** loop with extra options:

- If this is true, do this, then exit the **If**
- Else if (**elif**) this second logical statement is true, do this (then exit)
- If neither the "if" nor the "else if" are true, do the "else" (then exit)

You can add as many **elif** statements as you want, but remember that the robot will look at them in order, so once it gets to one statement that is true, it will not look at the rest of the "else if" arguments. Let's build a program that uses **Elif** and **Else** with our touch switches. Our program will back up and then turn left if bumper_8 is pressed, will back up and turn right if bumper_9 is pressed, and will continue to drive forward if neither is pressed. Remember that the bump switches are on the back of the robot so "forward" means "reverse!" Let's write our program, using the timer to control how long it will run:

1. Add a timer.start statement into your program
2. Add a **While Not** loop onto your program below the timer block. It should be "while the timer is less than 10 seconds, do this."
3. Start the drive train in reverse at -50% power.
4. Add an if statement that says "if bumper8 is pressed:"
5. Inside this if, add a statement using the drivetrain to drive straight at 50% power for 100mm, and another which uses the drivetrain to turn left at 50% for 90 degrees (left turn).
6. Add an **Elif** statement which reads "If bumper9 is pressed:"
7. Inside this elif, add a statement using the drivetrain to drive straight at 50% power for 100mm, and another which uses the drivetrain to turn right at 50% for 90 degrees (right turn).
8. Drag a drivetrain **Begin Driving** block and snap it into If block next to Else. Change "forward" to "reverse." The bump switches are on the back of the robot, so we will run in reverse instead of forward.
9. Below the while loop, turn off the drive train.

Your program should now look like this. Remember, the content in green is comments:

```
# VEX IQ Python-Project
import sys
import vexiq
import timer

#region config
motor_left = vexiq.Motor(1)
```

```
motor_right = vexiq.Motor(6, True) # Reverse Polarity
bumper_8 = vexiq.Bumper(8)
bumper_9 = vexiq.Bumper(9)
import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 176)
#endregion config

timer_ = timer.Timer() # Initialize timer

# main thread
timer_.start() # Start timer
while not timer_.elapsed_time() > 30: # while the timer is at less than 30
seconds
   if bumper_8.is_pressed(): # If bumper 8 is pressed do this
      dt.drive_until(50, 100)
      dt.turn_until(50, 90)
      vexiq.sound_wave(8) # exit If after this before else if (elif)
   elif bumper_9.is_pressed(): # If bumper 9 is pressed do this
      dt.drive_until(50, 100)
      dt.turn_until(-(50), 90)
      vexiq.sound_wave(12) # exit If after this before else if (elif)
   else:
      dt.drive(-(50)) # drive in reverse, bumper switches are on the back
      vexiq.sound_wave(1) # if the timer has run out, do this and exit
dt.off() # turn off the drivetrain
```

What do you think will happen? Let's test your program and find out:

Downloading and Testing Your Program

1. Plug your robot into your computer, and make sure the VEX IQ Brain is turned on.
2. Caution! Your program will run as soon as you download it, so make sure your robot is on the floor or someone is holding it. You do not want to drive it off the table!
3. Click **Run** to download your program to the robot. You should now see "Download Done" in a green bar at the bottom of the screen.
4. Unplug your robot, and set it on the floor.
5. On the Brain, use the up-arrow and down-arrow buttons to highlight the program you downloaded.
6. Making sure that you robot is in a safe area and no one has their hands or eyes near it (wear safety glasses), push the check-mark button on the Brain to start your program. Run it more than once if

you want to.

What Happened?

If you robot is built correctly and you entered the program above, you should have seen your robot drive in a straight line until one of the bumper switches touches. What happens after the bump switch on the left is activated? How about the one on the right?
You should have seen your robot back up and turn when it ran into an obstacle, and turn differently for each of the bumper switches. Why did this happen?

Here is what your robot program did:

1. Started
2. Read a block that told it to do everything in this loop until 30 seconds has gone by.
3. After starting the loop, the program goes on to the **If** loop, where it starts checking the conditions in the loop.
4. If bumper_8 is pressed, the robot will back up, then turn left. Once that is done, the program leaves the **If** loop and goes back to the **While** loop. If the timer has not finished, it runs the **If** statement again since that is the only code in the loop.
5. If bumper_9 is pressed, the robot backs up and turns right. Just like bumper_8, the program then exits the **If** block.
6. If neither bumper_8 nor bumper_9 is pressed, the program executes the code in the Else statement and then exits the **If** loop
7. Once the timer hits 10 seconds, the program stops.

You can see something in this program that you will notice a lot as you start programming. We want the drivetrain to start as soon as the program runs, but the only place we tell the robot to drive is in the Else statement of the If block. We did this because it is what the drivetrain will do if no sensors are triggered, in this case, the bumper switches. The computer program runs so fast that it doesn't matter that it checks the bumper switches before turning on the drivetrain for the first time.
If you want to see something fun, tape down one of the bumper switches and turn your robot on. What happened? If it turned in a jerky little circle then stopped played its music, you saw what we expected. Walk through the program yourself, pretending you are a robot, and see why this happens.

Activity Extension

Click **Save** before doing these extension activities
1. We suggested writing the code so that the robot backs up 100mm, then turns, then starts moving again. How would you change the code to try different ideas? Go ahead and make your changes, then test them and see what happens.

2. In the Play Wave function, there are lots of options. Try changing them and seeing what else you can play. Try some of the other VEX IQ Sound files. Can you get all of them to work at the end of this program?

3. You probably noticed that we go "reverse" to drive forward, and "forward" to go backwards. This is because the bumper switches are mounted on the back of the robot. Using what we learned about motors in the early units, how would you change the program or motor setup so that "forward" goes forward and "reverse" goes backwards? Hint: look at the motor settings. Apply your changes to the program and test it. Did it work?

Post-Activity Check-Up

1. Where would you use a timer loop?

2. In the **If** statement, we used a logical comparison to check if the bump switches had been pressed by the robot running into something. In your own words, what does this statement do if neither switch is pressed, only one switch is pressed, and if both are pressed. How would this be different if the statement said "Bumper_8 is pressed AND Bumper_9 is pressed?"

3. Why did we set the drivetrain to "reverse" to go backwards?

4. We used sound to tell us that the robot had stopped. What else could you use sound for on a robot?

8. Where's the Wall? Activity

Activity Overview

In this unit you will learn about:

1. How to use the Connect feature to see sensor values
2. Using the Distance Sensor
3. More LCD screen options
4. Using multiple sensors: bumper switches and distance sensors

Activity Preparation

This project uses the Autopilot Robot from Chapter 4 of the **VEX IQ Build Instructions**. You will need to build this robot before you start programming. See build steps 102-117 for instructions. This is the same robot you used in the last activity.
Before getting started make sure that you followed all the instructions in VEX IQ Controller Users Guide to configure your VEX IQ Brain, motors and sensors.
Prepare your robot:

1. Check to see that you have built the correct robot and that all motors and sensors are plugged into the correct ports
2. Check batteries
3. Install radios
4. Check to make sure the VEX IQ Brain is paired with Controller (if you are using the Controller)

Prepare RM Studio:

1. On the RM Studio project page click on **Create a New Project**
2. Click on **Target: VEX IQ** and **Language: Blockly, Flowol, Python or Controller Express**
3. Type your project name in the box. You might want to use the name of the activity as your project name
4. Click on **Create** and wait for the new project page to load
5. Plug your VEX IQ Brain into your computer using a USB A-to-micro cable (provided in the VEX kit)
6. Press the check mark button on the Brain to turn it on

7. Click on **Detect Sensors**. The panel on the right will now show two motors. If any other sensors were installed, those would show up, too
8. Check that the motors are plugged in and configured in the Device Monitor and Configuration panel
9. Change the name of motor_1 to motor_left, and motor_6 to motor_right
10. For "motor_right" click on the **Gear Icon**, select "Motor (reverse polarity)" in the Subtype box, and then click "**OK**."

Student Instructions

Getting Started

In this lesson, we will continue to use Robot Mesh Python to program our robot autonomously. We will learn more about sensors and outputs.

Set up the Drivetrain function

1. For "motor_right" click on the "v" and select "Motor (reverse polarity)"
2. On the far right, underneath the motor definitions, click on the gear button next to **Configure drivetrain here**.
3. Click the **Enabled** button.
4. In the **Left Motor** box pick "motor_left" and in the **Right Motor** box, pick "motor_right."
5. Click on **OK**.

Using Variables

In Python, you can store values in a variable. A variable is like box in which you store numbers or text strings. You can then use the variable in your program, even when you do not know its current value. It's very simple to do this in your program:

```
VariableName = <value>
```

For example, if you want the variable "Counter" to contain the value "15" you would use this in your program:

```
Counter = 15
```

If you want to store the current value of the timer in a variable, you could do something like this:

```
CurrentTimer = timer_1.elapsed_time
```

The CurrentTimer variable value will be fixed until you change it. In Python, you do do not need define what kind of value it is going to have. Python figures this out on its own. For example, both of these would be valid statements in Python:

```
NumberOfSheep = 10

NumberOfSheep = "ten"
```

In the first instance, the variable "NumberOfSheep" would be a number, and can be used as a number, but the second one has a string (or "text") value of "ten" which means you cannot do arithmetic with it. We don't often use text strings in programming robots, so your variables will likely include only numbers.

We will use variables to store the values of signals returned by sensors, among other purposes.

Using Distance Sensors to Find Objects

The Distance Sensor transmits ultra-sonic sound waves which then bounce off objects and return to the sensor. It measures the time it takes for the sound to get to an object and back and uses this to calculate how far away the object is in centimeters (cm). This is the same principal used by echo-locating owls, submarine sonar, and fish-finders on boats.

VEX IQ Distance Sensor

In our first exercise, we will use a distance sensor to look for an obstruction, and then keep the robot from running into it. We will use the timer routine to stop the program after running for a while.

1. Add a timer statement to your program, and another to start the timer running.
2. We want to use a **while** statement to run the code until the timer runs out. Let's start with "while" the timer is less than 15 seconds.
3. Start the robot moving forward at 50% using the drivetrain method.
4. Set a variable called "DistanceToObstacle" to the current value of the range finder using the "distance_10" function. See box above to learn about variables in Python.
5. Use an **if** statement to check if the current distance (stored in the variable "DistanceToObstacle") is less than 20 centimeters. If it is, do this:
   1. Drive backwards for 100mm (about 4 inches)
   2. Turn 135 degrees in either direction.
   3. Write the current value of "DistanceToObstacle" to the LCD display on the VEX IQ Brain. See box below for information on writing things to the Brain's display.
6. After the if block, turn off the drivetrain and play a sound effect.

Your program should now look like something like this. For clarity, we have removed configuration statements for sensors that we are not using:

```
# VEX IQ Python-Project
import sys
import vexiq
import timer

#region config
motor_left = vexiq.Motor(1)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
distance_sensor = vexiq.DistanceSensor(4, vexiq.UNIT_CM)
import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 166)
#endregion config

DistanceToObstacle = None

timer_1 = timer.Timer()

# main thread
timer_1.start()
```

```
while timer_1.elapsed_time() < 15:
        dt.drive(50)
        DistanceToObstacle = distance_sensor.distance()
        vexiq.lcd_write(DistanceToObstacle, 4)
        if DistanceToObstacle < 10:
                dt.drive_until(-(50), 100)
                dt.turn_until(-(50), 135)
dt.off()
vexiq.sound_wave(7)
```

What do you think will happen? Let's test your program and find out.

Downloading and Testing Your Program

1. Plug your robot into your computer, and make sure the VEX IQ Brain is turned on.
2. Caution! Your program will run as soon as you download it, so make sure your robot is on the floor or someone is holding it. You do not want to drive it off the table!
3. Click **Run** to download your program to the robot. You should now see "Download Done" in a green bar at the bottom of the screen.
4. Unplug your robot, and set it on the floor.
5. On the Brain, use the up-arrow and down-arrow buttons to highlight the program you downloaded.
6. Making sure that you robot is in a safe area and no one has their hands or eyes near it (wear safety glasses), push the check-mark button on the Brain to start your program. Run it more than once if you want to.

What Happened?

If you robot is built correctly and you entered the program above, you should have seen your robot drive in a straight line until it came within 10cm of an obstacle, then it should back up, turn, and start driving again. What happened with your robot?
Response:

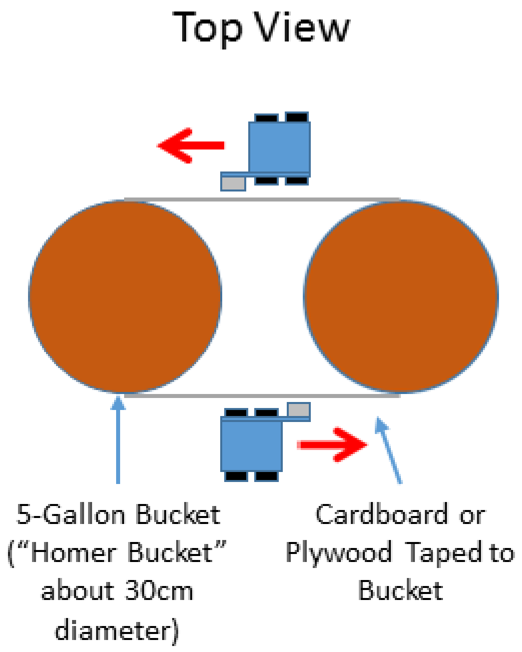
Robot Circle Racers

Let's use wall-following to play a game with your robots. This is our version of a robot racing game, where the robots run around a circular wall, using the Distance Sensor to guide them. It is based on pursuit racing in cycling, and on games from Parallax and the Sipna College of Engineering &

Technology. In cycling, the Individual Pursuit is a track race between two teams who start on opposite sides of the track, and then try to catch up with their opponent. You can find articles and video of pursuit on the Internet.

In our Robot Pursuit game, you will build a robot that will follow a rounded rectangular track. The first robot to catch up with or pass another robot that started on the other side of the track is the winner.

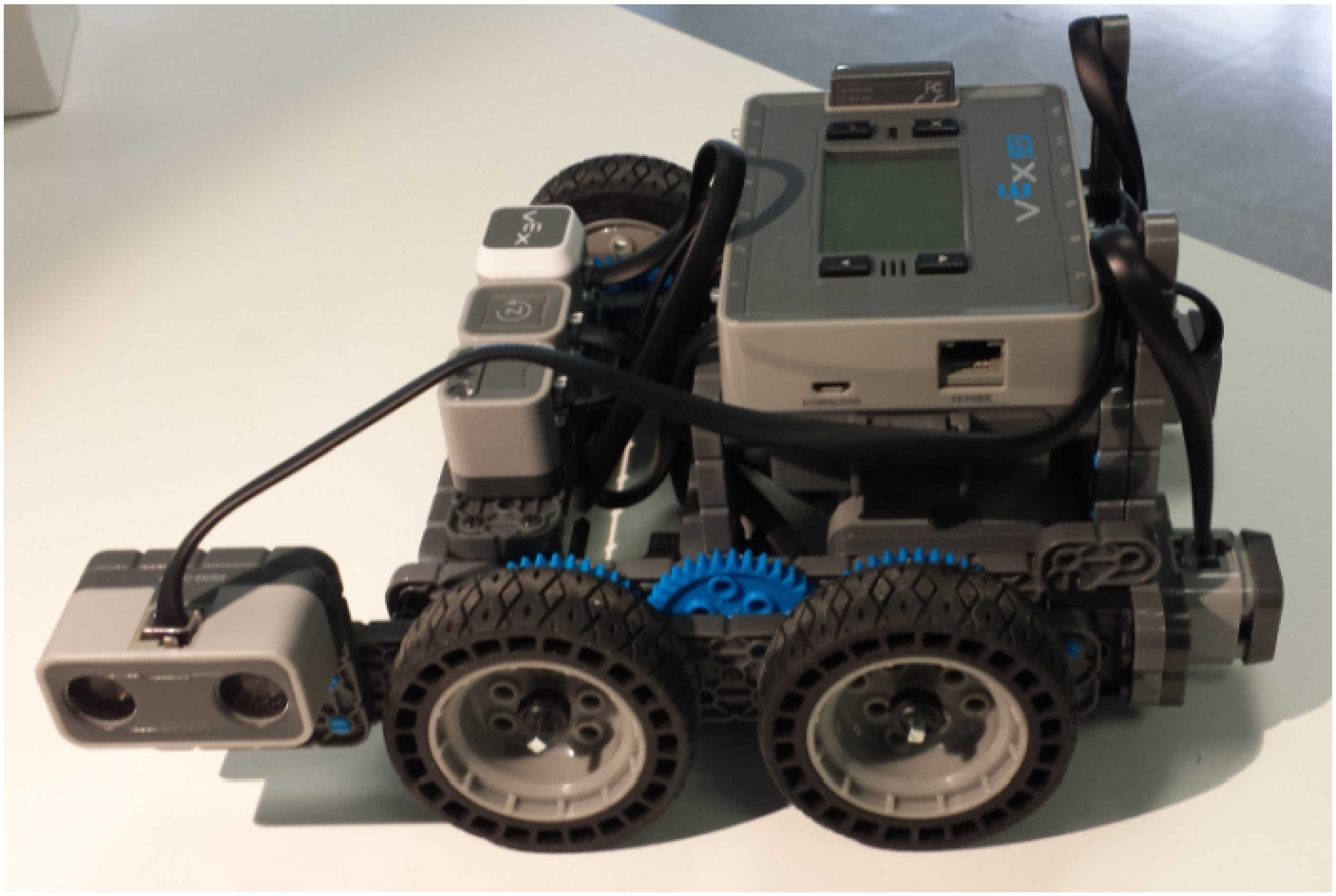
Here is a drawing of a "track" which you can make using a pair of paint buckets and some corrugated cardboard:


Top View
5-Gallon Bucket ("Homer Bucket" about 30cm diameter)
Cardboard or Plywood Taped to Bucket

To do this activity, you will need to make a change to your robot so that the Distance Sensor faces to the side and not the front of the robot. Here is what you need to do:

1. Remove both wheels on the left side of your robot.
2. Replace the 2x12 beam (the side rail) with a 2x16 beam.
3. Reassemble the side of your robot.
4. Move the Distance Sensor from the front of the robot to the outside end of the long beam you attached. You may need to use one of the longer Smart Cables if yours is too short.

This is what your robot should now look like:



Robot Racer Rules

The task in Robot Racer is to drive along the straight side of the course, turn around the curved end, drive back along the other straight, and make the final turn around the other curved end.

1. The race is done counter-clockwise, with the left side of your robot facing the "track."
2. Two robots start at the same time, on opposite sides of the track.
3. When the referee gives the word, both teams will push the check-mark button on their VEX IQ Brain to start their Python program.
4. The race is over when one robot completes three laps, or when one of the robots either passes or touches the robot ahead of it.
5. If neither robot can finish a complete lap, the race is called off and both teams will have a chance to change their program. The race is then re-started.
6. If one of the robots touches the wall during the first lap, the teams will get a programming break, and the race will be re-run.
7. After the first lap is complete, the race is official. If a robot goes hopelessly off-course or touches

the wall after the first lap, it is disqualified and the other robot is declared the winner.

Programming the Racing Robot

Since you know the wall is to the robot's left, and all turns will be to the left, you can program this with only one Distance Sensor. Part of what you are going to learn here is about proportional responses to errors. Later on, this leads to a robotics programming concept called PID, but for now, we will take a simpler route.

Thinking about the problem, here is one approach to its solution. You will probably think of a better way, but this will get you started. Using the Drivetrain feature:

1. Let's use the timer feature to control how long we want the robot to run. During testing, we checked to see how far our robot could go in 15 seconds to see how good our program was.
2. We want the robot to stay as close to the wall as possible Once the timer is started, check the Distance Sensor value.
3. This is where we use the "proportional" concept. The farther we are off-track, the larger the changes we will make to our path:
    1. If the robot is really close to the wall (less than 5 cm, or about 2 inches), move a little farther away.
    2. If it is the right distance from the wall (from 5-11 cm), keep going straight, the distance is fine.
    3. If the distance is farther (12-15 cm) make a small turn towards the wall and keep going.
    4. If the distance is even farther (16-24 cm) make a big turn towards the wall and keep going.
    5. If the distance is farthest (equal to or more than 25cm) make a bigger turn towards the wall and keep going.
4. We will use an **if - else if** logic block to program this logic.
5. We will keep checking the distance and setting the speed until the timer runs out.
6. At the end of the timer, we will turn off the drivetrain and sound a car alarm.

Once the code is entered in Python, you can experiment with three factors to adjust your robot:

- The settings for the Distance sensor
- How much the robot turns based on these distance readings
- How fast the robot is driving.

This kind of program usually takes some trial and error, because all robots are slightly different, even when built to the same plan. This Python program was written to drive around a 5-gallon paint can (about 30cm in diameter, a "Homer Bucket") and should get you started. The floor surface makes a big difference, and you might have to make adjustments if you are running on carpet. We tested this on a tile floor. Adjusting the "if" statements can optimize your robot to fit your circumstances, or to do better than this demo program:

```
# VEX IQ Python-Project
import sys
import vexiq
import timer

#region config
motor_left = vexiq.Motor(1)
Distance_Sensor = vexiq.DistanceSensor(4, vexiq.UNIT_CM)
touch_led_5 = vexiq.TouchLed(5)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
bumper_8 = vexiq.Bumper(8)
bumper_9 = vexiq.Bumper(9)
import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 166)
#endregion config

DistanceToObstacle = None

timer_1 = timer.Timer()

# main thread
timer_1.start()
touch_led_5.color(100,100,100)
while timer_1.elapsed_time() < 120: # run for 2 minutes
   DistanceToObstacle = Distance_Sensor.distance() # set variable
DistanceToObstacle to current distance
   vexiq.lcd_write(DistanceToObstacle, 3) # write distance to LCD display on
Brain
   if DistanceToObstacle < 5: # too close to wall
      dt.turn_until(-(50), 5)
      dt.drive(50)
      touch_led_5.color(0,0,60)
   elif DistanceToObstacle < 12: # just right
      dt.drive(50)
      touch_led_5.color(0,60,0)
   elif DistanceToObstacle < 16: # a little too far from wall
```

```
      dt.turn_until(50, 10)
      dt.drive(50)
      touch_led_5.color(100,60,100)
   elif DistanceToObstacle < 25: # too far from wall
      dt.turn_until(40, 15)
      dt.drive(50)
      touch_led_5.color(60,0,0)
   else: # much too far from wall, over 25 cm
      dt.turn_until(40, 20)
      dt.drive(50)
      touch_led_5.color(100,0,0)
   sys.sleep(0.2) # pause for .2 seconds
dt.off() # turn off drivetrain
vexiq.sound_wave(7) # play sound effect
```

You will see three more things in the program:

1. We added a short Sleep statement to give the robot time to respond to the commands to drive.
2. Right after we read the Distance Sensor, we write that value to the LCD display on the VEX IQ Brain.
3. We are using the Touch Sensor to give us an idea of how close we are to the right distance by changing the color. Blue means "too close," green means "on target," and various shades of red and pink show how far away from the wall the robot is. We will use the Touch Sensor more in the next unit.

The LCD display value and Touch Sensor color don't change the way the program works, but they give you tools to debug and improve your robot's performance.
Go ahead and build your program, download it and test it. At first, you might want to test it by running it along a wall or other long, straight object. We tested ours by having it run around a single bucket and a cardboard box.
What did you robot do? Does it follow a curved "wall" pretty well? If yours works like our test robot, it drives a short distance, stops and turns, and then continues doing that all the way around.

Activity Extensions

Click **Save** before doing these extension activities
1. The robot movement looks jerky because the Drivetrain function can only either turn OR move forward, but it cannot turn and move at the same time. What would happen if we used the Motor functions directly to smooth this out? It would take more programming, but it might work better. Try entering this program on your robot and test it. Is it smoother than the first program? Experiment with

the settings:

```
# VEX IQ Python-Project
import sys
import vexiq
import timer

#region config
motor_left = vexiq.Motor(1)
Distance_Sensor = vexiq.DistanceSensor(4, vexiq.UNIT_CM)
touch_led_5 = vexiq.TouchLed(5)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
bumper_8 = vexiq.Bumper(8)
bumper_9 = vexiq.Bumper(9)
#endregion config

SleepTime = None

DistanceToObstacle = None

timer_1 = timer.Timer()

# main thread
timer_1.start()
touch_led_5.color(100,100,100)
SleepTime = 0.2
while not timer_1.elapsed_time() >= 120:
   DistanceToObstacle = Distance_Sensor.distance()
   vexiq.lcd_write(DistanceToObstacle, 4)
   if DistanceToObstacle < 5:
      motor_left.run(100)
      motor_right.run(75)
      touch_led_5.color(0,0,60)
   elif DistanceToObstacle < 10:
      motor_left.run(100)
      motor_right.run(100)
      touch_led_5.color(0,60,0)
   elif DistanceToObstacle < 16:
      motor_left.run(65)
      motor_right.run(100)
      touch_led_5.color(100,60,100)
   elif DistanceToObstacle < 30:
      motor_left.run(65)
      motor_right.run(100)
      touch_led_5.color(60,0,0)
   else:
      motor_left.run(20)
      motor_right.run(100)
      touch_led_5.color(100,0,0)
   sys.sleep(SleepTime)
vexiq.sound_wave(7)
```

Response:
1. Using either program (the Drivetrain version or Motor version), try changing the If-Elseif-Else values. There are five different turn commands in each of these programs, could you write the code with only three? Would it be faster with seven? Try changing some things and see what makes your robot faster, smoother, and more effective.

2. Conduct a classroom tournament, playing the Robot Race Game. Work with your teacher to either hold a round-robin or elimination-bracket tournament. Were some robots much faster and more accurate than others?
Response:

Post-Activity Check-Up

1. Without writing the program, tell how you would change the robot and program to follow a race track with walls on the left, right and ahead.

2. How do you set the colors for the Touch Sensor?

3. Is it better to slow down when turning to follow a wall, or is it better to stay at full speed? Why?

4. What is the range of the Distance Sensor? What was the highest value you saw on the LCD screen when you were testing your robot?

5. Only answer this if you did the Activity Extension program. In your own words, which of these two programs are smoother and faster than the other? Why do you think this is?

9. Steering Using Gyros Activity

Activity Overview

In this unit you will learn about:

1. How to use the Gyro Sensor to make your robot complete a precise turn
2. How to use the Gyro Sensor to make your robot drive in a straight line
3. How to write your own function
4. Using comments in your code to make it more readable.

Activity Preparation

This project uses the Autopilot Robot from Chapter 4 of the **VEX IQ Build Instructions**. You will need to build this robot before you start programming. See build steps 102-117 for instructions. This is the same robot you used in the last activity. If you have the Distance Sensor mounted in the wall-following position, it will still work. You do not need to move the Distance Sensor before completing these activities.
Before getting started make sure that you followed all the instructions in VEX IQ Controller Users Guide to configure your VEX IQ Brain, motors and sensors.
Prepare your robot:

1. Check to see that you have built the correct robot and that all motors and sensors are plugged into the correct ports
2. Check batteries
3. Install radios
4. Check to make sure the VEX IQ Brain is paired with Controller (if you are using the Controller)

Prepare RM Studio:

1. On the RM Studio project page click on **Create a New Project**
2. Click on **Target: VEX IQ** and **Language: Blockly, Flowol, Python or Controller Express**
3. Type your project name in the box. You might want to use the name of the activity as your project name
4. Click on **Create** and wait for the new project page to load
5. Plug your VEX IQ Brain into your computer using a USB A-to-micro cable (provided in the VEX kit)

6. Press the check mark button on the Brain to turn it on
7. Click on **Detect Sensors**. The panel on the right will now show two motors. If any other sensors were installed, those would show up, too
8. Check that the motors are plugged in and configured in the Device Monitor and Configuration panel
9. Change the name of motor_1 to motor_left, and motor_6 to motor_right
10. For "motor_right" click on the **Gear Icon**, select "Motor (reverse polarity)" in the Subtype box, and then click "**OK**."

Student Instructions

Getting Started

In this lesson, we will continue to use Robot Mesh Blockly to program our robot autonomously. We will learn more about sensors and program design.

Set up the Drivetrain function

1. For "motor_right" click on the "v" and select "Motor (reverse polarity)"
2. On the far right, underneath the motor definitions, click on the gear button next to **Configure drivetrain here**.
3. Click the **Enabled** button.
4. In the **Left Motor** box pick "motor_left" and in the **Right Motor** box, pick "motor_right."
5. Click on **OK**.

Using Gyroscopic (Gyro) Sensors to Control Movement

The Gyro is an electronic part that can tell your program which way a robot, robot arm, or other component is facing. At the beginning of a program you "calibrate" the sensor with a special command, which sets it to a value near zero. If your Gyro readings change you can detect that in your program and change your robot's direction. As your robot turns right from its starting direction, the Gyro reading will be negative (less than zero) and positive (more than zero) if it turns left.

VEX IQ Gyro

In our first exercise, we will use a Gyro to help us make an accurate 90-degree turn.

The logic of our program is this:

1. Calibrate the Gyro. The solid state gyro on a VEX IQ robot needs to be "calibrated" before it works. The way you do this is add a gyro calibrate statement followed by a timer. We recommend setting the timer for 5-10 seconds.
2. Set the turn angle and display this in LCD row 1 of the VEX IQ Brain. The Gyro thinks "right" is negative and "left" is positive, so to make a 90-degree turn to the left, we would set this to "90," and to make a 90-degree turn to the right it would be set to "-90."
3. Drive forward 500 mm.
4. Drive forward, read the Gyro, then make adjustments.
5. We will use a **while** block to make the robot turn. The **while** statement will say why the current value of the Gyro is between -90 and +90 degrees, keep turning. When the Gyro angle is larger than the degrees we want to turn, the program will stop running the **while** block. Inside the block:
    1. The **if else** checks to see if the angle is negative or positive.
    2. If the angle is negative, the robot will start a right hand turn, and keep turning, or else it will turn left.
    3. We will display the gyro angle on line 4 of the Brain's LCD display.
6. Drive straight forward for 500 mm.
7. Sleep for 10 seconds to give the programmer a chance to see the values on the display.

Here is an example of what your first Gyro program might look like. Compare the steps above to this program and see if you can tell what the program is doing. Don't worry about the "math.fabs" function yet, we will go through that soon:

```
# VEX IQ Python-Project
```

```
import vexiq
import sys
import math

#region config
motor_left = vexiq.Motor(1)
distance_2 = vexiq.DistanceSensor(2, vexiq.UNIT_CM)
gyro_4 = vexiq.Gyro(4)
touch_led_5 = vexiq.TouchLed(5)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
bumper_8 = vexiq.Bumper(8)
bumper_9 = vexiq.Bumper(9)

import drivetrain
dt = drivetrain.Drivetrain(motor_left, motor_right, 200, 176)
#endregion config

GyroValue = None
AngleTurn = None

# main thread
gyro_4.calibrate()
sys.sleep(6)
GyroValue = 0

# Set AngleTurn to -90 for left, 90 for right turns
AngleTurn = -90 # AngleTurn is the variable we use to store the target angle
for the turn
vexiq.lcd_write(GyroValue, 1) # write the starting value of the Gyro to line
1 on the Brain
vexiq.lcd_write(AngleTurn, 2) # write the target turn angle to line 2 on the
Brain
vexiq.lcd_write(GyroValue, 4) # write the current value of the Gyro to line
4 on the Brain
dt.drive_until(50, 500)
GyroValue = gyro_4.angle()

# Gyro will start out around 0. Keep turning until the absolute value of
GyroValue is > 90
while math.fabs(GyroValue) <= math.fabs(AngleTurn):
        vexiq.lcd_write(GyroValue, 3)
        GyroValue = gyro_4.angle()
        vexiq.lcd_write(GyroValue, 4)
        # If angle is <0 turn left, otherwise right
        if AngleTurn < 0:
                dt.turn(20)
        else:
                dt.turn(-(20))
dt.drive_until(100, 500)
dt.brake()
sys.sleep(5)
```

What do you think will happen? Let's test your program and find out:

Downloading and Testing Your Program

1. Plug your robot into your computer, and make sure the VEX IQ Brain is turned on.
2. Caution! Your program will run as soon as you download it, so make sure your robot is on the floor or someone is holding it. You do not want to drive it off the table!
3. Click **Run** to download your program to the robot. You should now see "Download Done" in a green bar at the bottom of the screen.
4. Unplug your robot, and set it on the floor.
5. On the Brain, use the up-arrow and down-arrow buttons to highlight the program you downloaded.
6. Making sure that you robot is in a safe area and no one has their hands or eyes near it (wear safety glasses), push the check-mark button on the Brain to start your program. Run it more than once if you want to.

What Happened?

If you robot is built correctly and you entered the program above, you should have seen your robot drive in a straight line for 1000 mm, then it should stop, turn, and drive another 1000mm. What happened with your robot? Did your robot make a perfect turn? If it did not, try adjusting the wheelbase setting in the Drivetrain configuration, increasing the value if your robot did not turn far enough, and reducing it if it turned too far.
Response:

Math Functions in Python

Python includes a math functions library that gives you some powerful tools to manipulate numbers.

You can learn about them on the Python website at https://docs.python.org/2/library/math.html. Remember, that our Python is built using version 2.7 of the language, and not 3.x. Python 3 has significant changes from Python 2, and not everything works the same in the two versions. Because of the small amount of memory in a VEX IQ Brain, we chose to stick with version 2.7 for now.
To use a Python math function, we call functions in the math library, which is called "math." When we used the "math.fabs" function, the compiler automatically added "import math" to our program, to bring in the math library.
Math.fabs(x) returns the absolute value of the variable "x." In math class, you might see absolute number written this way: |x|. An absolute number is a number's distances from zero on a number line. This means that |-3| is the same as |3| since both are the same distance from zero, which is 3. So |-5.13| and |5.13| are the same and both are 5.13.
Why did we use the absolute value function in our program? It's because when we calibrated the Gyro we know that it will have a starting value of about zero. Sometimes, though, the Gyro might start with a very small positive number (such as .332) or a small negative number (such as -1.3). Those small non-zero numbers might cause problem in our logic. Here is what we are doing, in English:

1. Pick our turn angle. Let's assume that we want to turn 90 degrees to the right, which is -90 for our desired turn, stored in variable AngleTurn.
2. Check to see if the current Gyro value is negative, but not yet -90.
3. If our Gyro value starts at -1.1, our check "is current angle less than -90." Since -1.1 is more than -90 (remember, the computer only compares numbers and the small negative number -1.1 is MORE than -90 degrees).
4. If we use absolute value, though, when we do the test, we find that |-1.1| is LESS than |-90|, because the absolute value of -1.1 is 1.1, and of -90 is 90, so the current angle of 1.1 is less than 90, so we continue our turn to the right.

This can sound confusing at first, but you might try changing your program to get rid of the math.fabs function and see what it would take to work. We believe this is simpler than the other choices.

Activity Extensions

Click **Save** before doing these extension activities.
It is great that your robot can make a single turn, but how about making it do a little more?
1. Try some different angles. Try big numbers (like 200), small numbers (like 5) and don't forget to use both negative and positive values.

2. Modify your program to make two turns, one to the right, one to the left. How did it work?

Using Gyro Sensors to Drive Straight

Now that you have learned to use the Gyro to make a clean turn, let's explore how to use the Gyros to keep your robot on track when it is moving straight.

Calibrating and Zeroing the Gyro

Gyros should always start at about zero degrees after calibration, but this exact number can be a little above or below zero. In Blockly, there is no way to set the Gyro to 0 degrees in the program so we adjust the Gyro readings in the program to take this into account. For example, if the Gyro reading right after calibration is -2.4 degrees, we will subtract this from all Gyro readings in the program. At the beginning, we will subtract -2.4 degrees from the current reading of -2.4 degrees, which gives us an adjusted Gyro reading of 0 degrees. If the robot has turned 90 degrees to the right, the raw gyro reading would be -92.4 degrees, but when we subtract 2.4 degrees, the reading will be 90.

Touch LED

Functions

To make this easier in the program we will define a function to look up the Gyro value, subtract the correction, and give this value to the program. The function has a name, which is called by the main part of the program. In this case, we are creating a function called "GetCurrentGyroAngle" which you can see below. A function (sometimes called a subroutine) is a block of code that can be called from one or more other places in a program. It can either return values to the main program as we do below, or it can run some code and return to the main program. You can use the Function block to create something that you will use over and over, or sometimes just to isolate a complex or "wordy" piece of code that you already know works. You improve the readability of your code by using subroutines and functions.

Here is what our program does

1. Set the touch sensor color to WHITE.
2. Calibrate the gyro
3. Create the variable GyroAdjustment to the initial value of the Gyro. This adjustment is derived by multiplying the initial value of the Gyro to itself multiplied by -1. This will make more sense later!
4. Write the current Gyro value to the LCD in row 5, by looking up the value in the "Return Current Gyro Angle" function.
5. Start the Timer
6. Turn the drive motors on.
7. Start a While loop which will continue until the timer runs out.
8. Fetch the Gyro angle from the function "Return Current Gyro Angle."
9. Decide whether the robot needs to turn right to correct its course (where the Gyro angle is greater than 1), or left (where the Gyro angle is less that -1). For a left turn, set the Touch Sensor to "red," and for a right turn, set the Touch Sensor to "green."
10. Either left or right, start a Repeat Until loop which will make the robot start a turn in the correct position to correct its course.
11. Once the timer is done, exit the Repeat Until loop for the timer.
12. Run the motors for a short time, and then shut off.

Here is what your code might look like:

```
import vexiq
import sys
import timer

#region config
motor_left  = vexiq.Motor(1)
distance_2  = vexiq.DistanceSensor(2, vexiq.UNIT_CM)
gyro_4      = vexiq.Gyro(4)
touch_led_5 = vexiq.TouchLed(5)
```

```
motor_right = vexiq.Motor(6, True) # Reverse Polarity
bumper_8    = vexiq.Bumper(8)
bumper_9    = vexiq.Bumper(9)

import drivetrain
dt          = drivetrain.Drivetrain(motor_left, motor_right, 200, 176)
#endregion config

GyroValue = None
GyroAdjustment = None

def GetCurrentGyroAngle():
  global GyroValue, GyroAdjustment
  # Return current gyro angle and write value to LCD
  GyroValue = gyro_4.angle() + GyroAdjustment
  vexiq.lcd_write(GyroValue, 3)
  return GyroValue

timer_1 = timer.Timer()

# main thread
vexiq.lcd_write('Calibrating Gyro', 1)
touch_led_5.color(100,100,100)
gyro_4.calibrate()
sys.sleep(10)
GyroAdjustment = gyro_4.angle() * -1
vexiq.lcd_write((GetCurrentGyroAngle()), 5)
timer_1.start()
vexiq.sound_wave(0)
while timer_1.elapsed_time() < 21:
    GyroValue = GetCurrentGyroAngle()
    if GyroValue < 0:
        touch_led_5.color(100,0,0)
        vexiq.lcd_write((GetCurrentGyroAngle()), 4)
        motor_left.run(50)
        motor_right.run(54)
        sys.sleep(.2)
    elif GyroValue > 0:
        touch_led_5.color(20,100,20)
        vexiq.lcd_write((GetCurrentGyroAngle()), 3)
        motor_left.run(54)
        motor_right.run(50)
        sys.sleep(.2)
    touch_led_5.color(100,100,100)
    motor_left.run(50)
    motor_right.run(50)
```

Download your code and see if your robot goes straight.

Post-Activity Checkup

1. What is a VEX IQ Gyro? The Gyro produces numbers, what do they mean? What are the lowest and highest readings the Gyro produces?

2. What should you do in your program to get the Gyro ready to go?

3. What does the math.fabs function do?

4. Why did we calculate the GyroAdjustment variable? How was it used in the program?

5. We have used three different ways of programming the robot to let the programmer/operator know how and what it is doing. Name at least two of them, and describe how you might use them.

6. What does a "While" loop do?

10. Mathematics for Robots Activity

Activity Overview

In this unit you will learn about:

1. The math functions available in Python
2. How to use math functions in your programming
3. The Print function
4. How to use Lists
5. Expanding on our use of Variables and Functions

Activity Preparation

This project uses the Autopilot Robot from Chapter 4 of the **VEX IQ Build Instructions**. You will need to build this robot before you start programming. See build steps 102-117 for instructions. This is the same robot you used in the last activity. If you have the Distance Sensor mounted in the wall-following position, it will still work. You do not need to move the Distance Sensor before completing these exercises.
Before getting started make sure that you followed all the instructions in VEX IQ Controller Users Guide to configure your VEX IQ Brain, motors and sensors.
Prepare your robot:

1. Check to see that you have built the correct robot and that all motors and sensors are plugged into the correct ports
2. Check batteries
3. Install radios
4. Check to make sure the VEX IQ Brain is paired with Controller (if you are using the Controller)

Prepare RM Studio:

1. On the RM Studio project page click on **Create a New Project**
2. Click on **Target: VEX IQ** and **Language: Blockly, Flowol, Python or Controller Express**
3. Type your project name in the box. You might want to use the name of the activity as your project name
4. Click on **Create** and wait for the new project page to load

5. Plug your VEX IQ Brain into your computer using a USB A-to-micro cable (provided in the VEX kit)
6. Press the check mark button on the Brain to turn it on
7. Click on **Detect Sensors**. The panel on the right will now show two motors. If any other sensors were installed, those would show up, too
8. Check that the motors are plugged in and configured in the Device Monitor and Configuration panel
9. Change the name of motor_1 to motor_left, and motor_6 to motor_right
10. For "motor_right" click on the **Gear Icon**, select "Motor (reverse polarity)" in the Subtype box, and then click "**OK**."

Student Instructions

Getting Started

In this lesson, we will explore the use of some of some of Robot Mesh Python's math functions.

Math in Python

Python includes the Python math library, which can do far more than is in the scope for these introductory lessons. In this activity we will explore a few math functions and show how they can be used. We encourage you to explore the others when you have time.
One of the math functions in Python generates a random number. We are going to use this function to roll imaginary dice and use the results of the dice roll to decide which way to have your robot move. To do this we will use these functions:
Random integer from X **to** Y
This function generates a random number between X and Y. In our case we are going to use "1" and "6" because those are the numbers that rolling a standard 6-sided die produces. We will then assign the random number to a variable.
Set variable value
We have used this before, but we are going to start using it more in future programs. Before using a variable, it is best to give it an initial value at the beginning of the program, along with a comment about what it is used for.
List Insert
In computer programming, you will frequently want to save a bunch of values while the program is

running, and in Python you use lists for these. For example, if you have 30 students you might want to have a list of them called Students. Python allows you to give them all the same name (Students) with a number for each, starting with 0. This is an example:
Students[0] = Gajjar
Students[1] = Ablett
Students[2] = Johnson
... and so on. In our program below you will see us building a list of the dice rolls we generate while the code is running.
List Get
This is a companion to **List Insert**, and retrieves the values stored by the insert command.
Print
This command prints information on your computer screen in the Output Window at the bottom of the programming area, while your program is running, as long as the programming cable is plugged in. Sometimes, it is useful to do this to see more about what is going on with your program than will fit on the LCD screen, and it does not disappear when the program stops. We will use this to show you the dice rolls as they happen. Print is not like your other programming commands, though, as it requires the robot to "talk to" the programming computer. You can test this by putting your robot up on a block and letting it run without the wheels touching the table or floor, with the programming cable still plugged in. We frequently test our programs like this before putting them on the floor.
Modulo
In this program we want to check the dice roll to see if it is odd or even. We do this using the % symbol, which is the Python modulo operator, meaning when its arguments are numbers, it divides the first by the second and returns the remainder. For example, "14 % 3" has the value of "2" since 14 divided by 3 is 4 with a remainder of 2. Another example "4 % 2" has the value of "0" since 4 divided by 2 is 2 with no remainder. The return value of the modulo operator is the remainder value.
We will be using the Modulo to tell us whether a dice roll is positive or negative.
Our Program
Here is a picture of our program. Enter this in Python. Here is what the main program should look like:

```
import vexiq
import timer
import sys
import random

#region config
```

```
motor_left  = vexiq.Motor(1)
distance_2  = vexiq.DistanceSensor(2, vexiq.UNIT_CM)
gyro_4      = vexiq.Gyro(4)
touch_led_5 = vexiq.TouchLed(5)
motor_right = vexiq.Motor(6, True) # Reverse Polarity
bumper_8    = vexiq.Bumper(8)
bumper_9    = vexiq.Bumper(9)

import drivetrain
dt          = drivetrain.Drivetrain(motor_left, motor_right, 200, 176)
#endregion config

DieRoll = None
NumberOfRolls = None
RollsList = None
TotalDiceRolls = None
Die1 = None
Die2 = None

timer_ = timer.Timer()

"""Describe this function...
"""
def RollDice():
  global Die1, Die2, DieRoll, NumberOfRolls, RollsList
  # The RollDice function rolls 2 "dice" and adds them
  # together, returning DieRoll. It also counts the
  # number of rolls in NumberOfRolls, and adds the
  # DieRoll value to RollList - a complete list of
  # all rolls.
  Die1 = random.randint(1, 6)
  Die2 = random.randint(1, 6)
  DieRoll = Die1 + Die2
  vexiq.lcd_write(DieRoll, 3)
  NumberOfRolls = NumberOfRolls + 1
  RollsList.append(DieRoll)
  print (RollsList[-1])
  return DieRoll


# main thread
# Initialize variables and start the timer
```

```
timer_1.start()
NumberOfRolls = 0
RollsList = []
while timer_1.elapsed_time() < 20:
  # Call the function RollDice and return this value
  DieRoll = RollDice()
  if DieRoll == 7:
    # If the roll is 7, go straight
    touch_led_5.color(0,0,60)
    motor_left.run(40, 200, False)
    motor_right.run(40)
    sys.wait_for(motor_left.reached_target_or_stalled)
  elif DieRoll % 2 == 0:
    # If the roll is an even number, turn right
    touch_led_5.color(20,100,20)
    motor_left.run(50, 200, False)
    motor_right.run(25)
    sys.wait_for(motor_left.reached_target_or_stalled)
  else:
    # Otherwise, turn right
    touch_led_5.color(100,0,0)
    motor_right.run(50, 200, False)
    motor_left.run(25)
    sys.wait_for(motor_right.reached_target_or_stalled)
  print 'last roll:'
  print (RollsList[0])
TotalDiceRolls = sum(RollsList)
print 'Average dice roll:'
print (float(TotalDiceRolls) / NumberOfRolls)
```

Notes on what the function RollDice does:

1. This function uses the Random Integer function to generate two numbers, each of which is between 1 and 6.
2. The next line sets the variable DieRoll to the sum of the two dice rolls.
3. We write the DieRoll value to the LCD, and increment (which means, "increase by one" in this case) the counter NumberOfRolls.
4. We add the current DieRoll to the list of all dice rolls, RollsList, using the In List Insert block, and print the most recent value to the system screen. This only works if the robot is running while still tethered. Finally, the function returns DieRoll to the point in the program where the function was

called.

In the main program:

1. We start the timer.
2. We initialize the variable NumberOfRolls to 0 and create the RollsList.
3. We start a **While** loop which will run for a certain number of seconds based on the timer.
4. We call the RollDice function, and assign its return value to the variable DieRoll.
5. We then start and **If Do** statement. We have written the program so that when the DieRoll is an even number, the robot will turn right, and when it is odd, the robot will turn left. Both of these use the math function Variable is Odd/Even. You should note that this function offers Prime/Whole/Positive/Negative/DivisibleBy as options in addition to Odd/Even.
6. We know that if you roll dice and write down the results you will get a "normal distribution" between 2 (the lowest possible roll of two dice) and 12 (the highest). If you do not know what a "normal distribution is, do not worry about it, but you could ask your teacher. The number in the center (called the "median") is 7, with 5 lower numbers possible (2,3,4,5,6) and 5 higher (8,9,10,11,12). Since there are six odd numbers in this set, and only five even numbers, when we roll a 7 we will drive straight.
7. Once again we will use the touch sensor colors to show you what is going on with your robot: green for a right turn, red for a left turn, and blue for straight ahead.
8. After the end of the **Repeat** loop, we create the variable TotalDiceRolls and set it the sum of all the rolls in the RollsList list. In other words, if there were 20 rolls, these would all be stored in the list RollsList, and we would use the math function "sum" to add them all together. We then divide TotalDiceRolls by the NumberOfRolls variable which counted the number of rolls we did. This gives us the average number rolled by our program, rounded to the nearest whole number.

What do you think will happen? Let's test your program and find out:

Downloading and Testing Your Program

1. Plug your robot into your computer, and make sure the VEX IQ Brain is turned on.
2. Caution! Your program will run as soon as you download it, so make sure your robot is on the floor or someone is holding it. You do not want to drive it off the table!
3. Click **Run** to download your program to the robot. You should now see "Download Done" in a green

bar at the bottom of the screen.

4. Unplug your robot, and set it on the floor.
5. On the Brain, use the up-arrow and down-arrow buttons to highlight the program you downloaded.
6. Making sure that you robot is in a safe area and no one has their hands or eyes near it (wear safety glasses), push the check-mark button on the Brain to start your program. Run it more than once if you want to.

What Happened?

Since your robot turns right on even numbers and red on odd, did you expect the turns to all cancel out and have your robot more or less in a straight line from where it started? Is this what you saw? What did your robot do? Run it 5 or more times, and write down what it did each time. Does your robot always go right or always go left?
For at least one run, put your robot "up on blocks" (we use the 3" Highrise cubes, but a lot of things would work) and download your program and let it run. Watch the Output Window in Python to see the **Print** statement writing the dice rolls.

Activity Extensions

Click **Save** before doing these extension activities.
It is great that your robot can make a single turn, but how about making it do a little more:
1. What if the robot went left for numbers less than 7 and to the right for numbers over 7? Change the code and test it. Does it behave differently?

2. Add more Print commands to watch variables and steps in the program. Watch the Output Window to see your changes.

Post-Activity Checkup

1. What is a Python list? How do we add items to lists, and how can we read them? Take a look at the User's Guide and explore some other options.

2. What does the Print command do, and how is it different from the LCD Write in Row command?

3. What does the Random Integer math function do?

4. The function could be shorter while still doing the exact same things it does now. Can you think of how to remove or combine steps to make it shorter?

5. Why does your robot not always end up in the exact same place?

6. If you ran the robot 100 times and added up all the times it went to the left and to the right, what would you expect to find? Why is this?

11. Controller Plus Programming Activity

Activity Overview

In this unit you will learn about:

1. How to combine Controller (joystick) commands with programming
2. Designing and programming on your own.

Activity Preparation

This project uses the Autopilot Robot from Chapter 4 of the **VEX IQ Build Instructions**. You will need to build this robot before you start programming. See build steps 102-117 for instructions. This is the same robot you used in the last activity. You should remove the Distance Sensor for this activity.
Before getting started make sure that you followed all the instructions in VEX IQ Controller Users Guide to configure your VEX IQ Brain, motors and sensors.
Prepare your robot:

1. Check to see that you have built the correct robot and that all motors and sensors are plugged into the correct ports
2. Check batteries
3. Install radios
4. Check to make sure the VEX IQ Brain is paired with Controller (if you are using the Controller)

Prepare RM Studio:

1. On the RM Studio project page click on **Create a New Project**
2. Click on **Target: VEX IQ** and **Language: Blockly, Flowol, Python or Controller Express**
3. Type your project name in the box. You might want to use the name of the activity as your project name
4. Click on **Create** and wait for the new project page to load
5. Plug your VEX IQ Brain into your computer using a USB A-to-micro cable (provided in the VEX kit)
6. Press the check mark button on the Brain to turn it on
7. Click on **Detect Sensors**. The panel on the right will now show two motors. If any other sensors were installed, those would show up, too

8. Check that the motors are plugged in and configured in the Device Monitor and Configuration panel
9. Change the name of motor_1 to motor_left, and motor_6 to motor_right
10. For "motor_right" click on the **Gear Icon**, select "Motor (reverse polarity)" in the Subtype box, and then click "**OK**."

Student Instructions

In this lesson, we will explore how to use the Controller and programming at the same time. To combine these, you will program your Autopilot robot to play Freeze Tag.

VEX IQ Controller

Basic Robot Freeze Tag

Freeze tag is a game in which the "It" person tries to touch someone else playing. When that person is touched, they have to "freeze" and not move until the "It" has tagged everyone playing. The robot version is similar. Here are the rules for Robot Freeze Tag:

1. Drivers will operate robots using the Controller.
2. Robots all have two bumper switches on the rear of the robot.
3. Robots play in a restricted area, determined by the Referee. Starting positions of the robots will also be set by the referee.
4. When a robot is tagged (one of its bumper switches is pressed by another robot), it is frozen.
5. A frozen robot may NOT respond to commands from the controller. The referee will be checking this, so make sure your program insures that your robot cannot respond to controller commands while frozen.
6. A frozen robot must flash a red light to show that it is frozen and out of the game.
7. Any robot may tag any other unfrozen robot.
8. The game may be played in any restricted area, such as a VEX IQ competition field or defined area

of classroom floor. Blue painter's tape can make a quick and simple arena. Any robot leaving the playing area is considered frozen, and can no longer play.

Programming Your Robot

You will need to program your robot's motors to respond commands sent from the Controller. As you remember from the Controller Express Unit, "AxisA" is on the left joystick, and is used like any other sensor. In this case, Python will read the value of AxisA (which can be from -100 to 100, from full reverse to full forward). The right joystick is "AxisD." The command to send Controller joystick information to the left and right drive motors, for example, is:

```
motor_left.run((joystick.axisA()))
motor_right.run((joystick.axisD()))
```

You will find the "position" button in the VEX IQ Sensors section of the Toolbox. This is also where you will find the "is pressed" button, which is how you can read the value of the Controller digital buttons. Make sure you add comments to your program! This is very important as you write bigger and bigger programs.

Programming for Basic Robot Freeze Tag

You will need to write this program on your own, but here are some tips:

1. Remember, you do not need a timer, the game goes until only one robot is NOT permanently frozen.
2. In our sample program, we used these features of Python. You can program this differently, but knowing what we did might help get you started:
    1. We used motor controls instead of a drivetrain.
    2. We used both bumper switches (as in, "If bumper 8 is touched, or bumper 9 is touched, the robot should be frozen")
    3. We used a Repeat loop to keep checking for controller commands and touches
    4. When the robot is tagged, we do not turn off the program, but we do stop the robot from moving.

Write your code, walk through it to see if looks like it will work, then give it a try!

Downloading and Testing Your Program

1. Plug your robot into your computer, and make sure the VEX IQ Brain is turned on.
2. Caution! Your program will run as soon as you download it, so make sure your robot is on the floor or someone is holding it. You do not want to drive it off the table!
3. Click **Run** to download your program to the robot. You should now see "Download Done" in a green bar at the bottom of the screen.
4. Unplug your robot, and set it on the floor.
5. On the Brain, use the up-arrow and down-arrow buttons to highlight the program you downloaded.
6. Making sure that you robot is in a safe area and no one has their hands or eyes near it (wear safety glasses), push the check-mark button on the Brain to start your program. Run it more than once if you want to.

What Happened?

Get a few robots going, and play the game according to the rules. How did it go? Did your program work the way you thought it would? If your program did not work the way you thought it would, make some changes and try again. When everyone in your class has a working robot, play one last game. The last moving robot is the winner!

Advanced Robot Freeze Tag

The advanced version of Robot Freeze Tag adds some features to make the game more interesting. Here are the rules for Advanced Robot Freeze Tag:

1. Drivers will operate robots using the Controller.
2. Robots all have two bumper switches on the rear of the robot.
3. Robots play in a restricted area, determined by the Referee. Starting positions of the robots will also be set by the referee.
4. While a robot is unfrozen, it must display a green light.
5. When a robot is tagged (one of its bumper switches is pressed by another robot), it will stop moving for 3 seconds, and display a red light and a short sound effect.
6. When a robot is frozen, no other robot can be within 30 cm (about a foot) of the frozen robot. The tagging robot must back off at least 30 cm from the frozen robot. This is to prevent a robot from

sitting behind a frozen robot and tagging it immediately after it turns back on.

7. After being frozen three times, the robot is permanently frozen, and is out of the game. A permanently frozen robot should flash red and white lights alternately. No other robot may interact with a permanently frozen robot.
8. A frozen robot may NOT respond to commands from the controller. The referee will be checking this, so make sure your program insures that your robot cannot respond to controller commands while either frozen or permanently frozen.
9. Any robot may tag any other unfrozen robot.

You will need to write this program on your own, but we can offer some tips:

1. Remember, you do not need a timer, the game goes until only one robot is NOT permanently frozen.
2. In our sample program, we used these features of Python. You can program this differently, but knowing what we did might help get you started:
   1. We set variables for the number of times the robot has already been tagged
   2. We used motor controls instead of a drivetrain.
   3. We used the Touch LED as the light
   4. We used both bumper switches (as in, "If bumper 8 is touched, or bumper 9 is touched, the robot should be frozen)
   5. We used a Repeat loop to keep checking for controller commands and touches
   6. When a robot gets its third tag, we do not turn off the program, but we do stop the robot from moving, and we also flash the red and white lights.

What do you think will happen? Play Advanced Robot Freeze Tag with some other robots and see how they work.

Activity Extension Ideas

Click **Save** before doing these extension activities.
Change the rules of the game, program your changes, and try it again.
1. Less or more than three freezes?

2. Rather than permanently freezing after 3, change it to a timer game and the winner is the robot with

the fewest tags within your time limit. How will you count and then display the tag counts?

3. What if you have to press both bumper switches at once to get a tag?

Post-Activity Checkup

1. What Python commands did you use to write your program?

2. In what ways is input from the Controller like sensor inputs?

3. How well did your robot drive? Was it easy to make the robot go in one direction? How would you change your robot's program to make the robot drive more easily?

4. In addition to the joysticks, the Controller has buttons you can program. Name at least one way you could program a button in this robot, or, if you programmed a Controller button already, how did you use it?

5. Could you use Controller buttons to change the color of the Touch Sensor LED? How would you program that?

12. Amazing Mazes Activity

Activity Overview

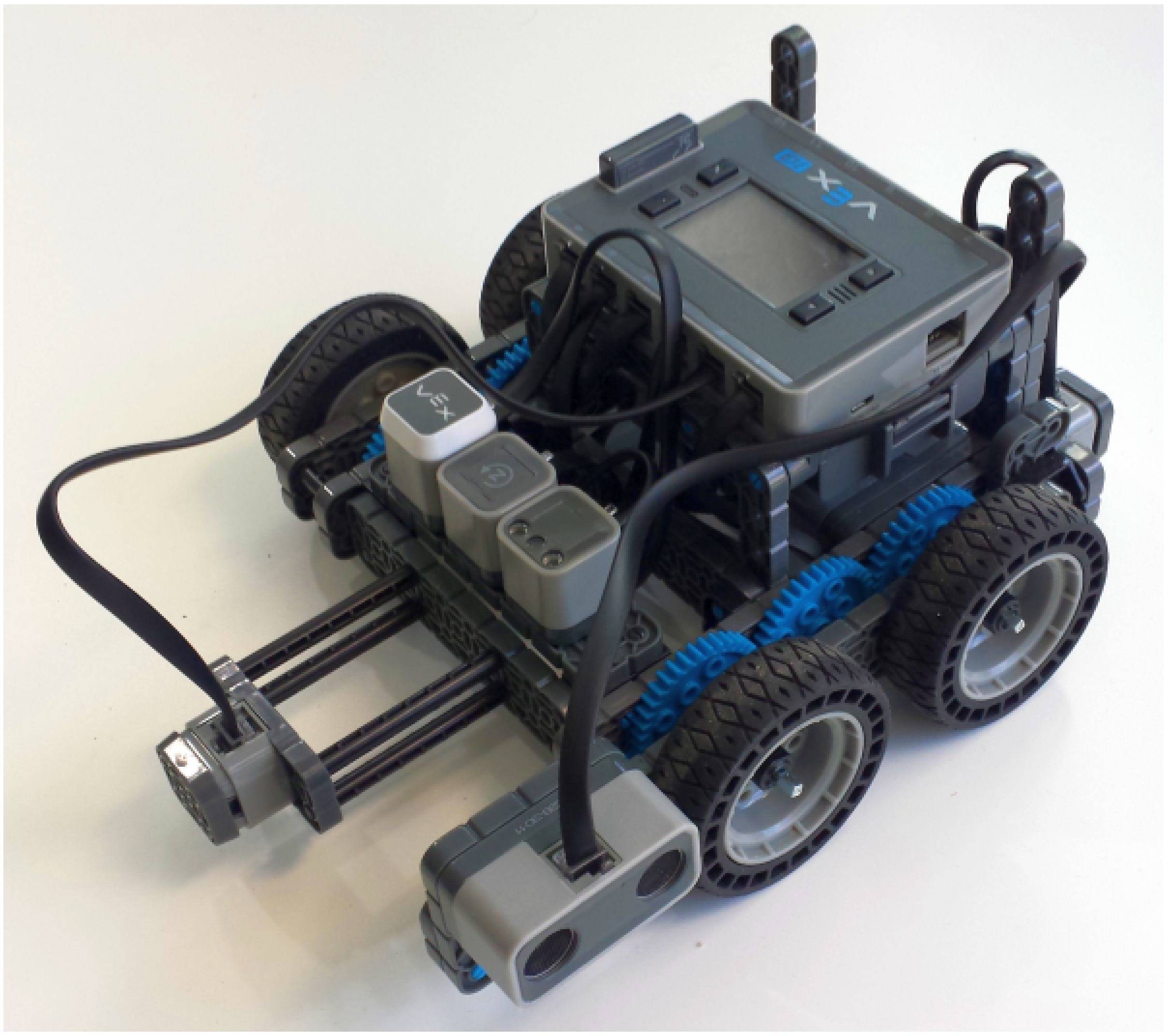
You have already learned:

- How to configure motors, outputs and sensors
- How to combine Controller (joystick) commands with programming
- How to autonomously program your robot to turn and go straight

Activity Preparation

This project uses the Autopilot Robot from Chapter 4 of the **VEX IQ Build Instructions**. You will need to build this robot before you start programming. See build steps 102-117 for instructions. This is the same robot you used in the last activity.

You will need to make two modifications to the robot for this activity. First, move the distance sensor to the left side of the robot, the same we did in the "Where's the Wall" activity (see Unit 8 Activity: Where’s the Wall for details). Additionally, we will move one of the bump sensors from the rear of the robot to the front, using four 2-inch (approximately 51mm) standoffs. Your robot should look like this when you have modified it:



Before getting started make sure that you followed all the instructions in VEX IQ Controller Users Guide to configure your VEX IQ Brain, motors and sensors.
Prepare your robot:

1. Check to see that you have built the correct robot and that all motors and sensors are plugged into the correct ports
2. Check batteries
3. Install radios
4. Check to make sure the VEX IQ Brain is paired with Controller (if you are using the Controller)

Prepare RM Studio:

1. On the RM Studio project page click on **Create a New Project**
2. Click on **Target: VEX IQ** and **Language: Blockly, Flowol, Python or Controller Express**
3. Type your project name in the box. You might want to use the name of the activity as your project

name

4. Click on **Create** and wait for the new project page to load
5. Plug your VEX IQ Brain into your computer using a USB A-to-micro cable (provided in the VEX kit)
6. Press the check mark button on the Brain to turn it on
7. Click on **Detect Sensors**. The panel on the right will now show two motors. If any other sensors were installed, those would show up, too
8. Check that the motors are plugged in and configured in the Device Monitor and Configuration panel
9. Change the name of motor_1 to motor_left, and motor_6 to motor_right
10. For "motor_right" click on the **Gear Icon**, select "Motor (reverse polarity)" in the Subtype box, and then click "**OK**."

Student Instructions

Robot Maze Running

In this activity, we will be programming your robot to navigate a maze without using the Controller.

Introduction to Mazes

A maze is a network of passages designed as a puzzle through which your robot has to navigate to the end. Since you are new at this, we encourage you to use a simple maze, where there is one path to the exit (or target square) and no loops. In other words, every possible branch dead-ends somewhere. There is a good article on mazes in Wikipedia, and if you do an Internet search on "robot mazes" you will find a lot of pictures and video.
Your robot will start in a defined location, and is done when it reaches the target. There are a few simple strategies, and ways to program maze-following robots. You are a pretty good programmer by now, so we will discuss how we solved the problem and give you some strategy ideas, but you will write your own program.

Maze-Solving Algorithms

"Algorithm" is a term used in math, engineering and programming, and simply means the steps or rules used to solve a problem. Since people have been solving mazes for at least 2,500 years, there is a lot of

information available on maze-solving algorithms. We want you build a perfect maze, one without loops, so that you can use the "follow the wall" algorithm. Your robot can simply follow either the right or left wall of the maze, and eventually get to the target spot. Since your robots have the distance sensor on the left side, following the left wall might make the most sense.
Following the wall is pretty simple:

1. Look to the left, and if you can make a left turn, do so.
2. If you cannot make a left turn, go straight.
3. If you cannot go straight, turn right. After turning right, go back to step 1.
4. Some people add, if you cannot turn left OR right, turn 180 degrees. If your maze is wide enough, two right turns equals one 180-degree turn.

Building Your Maze

You will need a maze with nice, wide passages for your VEX IQ robot. We tested with paths that were at least 12-16" (300-400mm) wide. These robots turn pretty wide and smaller passages did not work that well.
We tested several things for walls, and nearly any hard, fairly heavy material works. Construction timber (2x4 in the US, or 4x2 in Europe) cut to various lengths works very well, and you could use 2-sided carpet tape to stick them to the floor if they are sliding around. We have a VEX IQ field, so we also tried parts for two mazes using VEX IQ parts. The first one used beams mounted to the floor, but we were concerned that the ultrasonic distance sensors would not work because of all the holes, but, in fact, they worked well. It takes a lot of beams, though, and it took a long time to build. The second was to take the walls from a VEX IQ competition field, and turn them upside down. The holes in the tops of the walls can be aligned with the holes on the floor tiles to make maze walls. The two problems with this is that the tabs that stick out on one side tend to get tangled in the robots, and it takes a lot of wall pieces. If you do not have two competition fields this would be difficult.

Programming Your Robot

Before you start programming, write out your algorithm for your teacher. Be sure you can explain everything that can happen before you start writing your program.
If you are going to follow the left-hand wall, here are some things to think about:

1. You already wrote a program that follows a wall. Could you modify this program?
2. Some maze robots use more than one distance sensor, but if you only have one, are there other

sensors you could use in addition? We suggested moving a bump switch to the front of your robot, how could that help?
3. You have another bump sensor, how could adding that help your program?
4. Could the Touch LED light help you know what your robot is doing?

We suggest that you start by programming the robot above, but feel free to modify the sensor locations.

Downloading and Testing Your Program

1. Plug your robot into your computer, and make sure the VEX IQ Brain is turned on.
2. Caution! Your program will run as soon as you download it, so make sure your robot is on the floor or someone is holding it. You do not want to drive it off the table!
3. Click **Run** to download your program to the robot. You should now see "Download Done" in a green bar at the bottom of the screen.
4. Unplug your robot, and set it on the floor.
5. On the Brain, use the up-arrow and down-arrow buttons to highlight the program you downloaded.
6. Making sure that you robot is in a safe area and no one has their hands or eyes near it (wear safety glasses), push the check-mark button on the Brain to start your program. Run it more than once if you want to.

What Happened?

Run your robot through the maze. Did it get stuck anywhere? Were you able to "fix" your program so it finished successfully? How long did the run take? Does your robot always do exactly the same path, or is it slightly different?

Activity Extension Ideas

Click **Save** before doing these extension activities.
Here is an activity modification to help you build a better maze-solving robot, plus add some competition the activity. Do a maze-running competition between the robots in your class. The robot that solves the maze the fastest is the winner.

1. Use the same maze you already used.
2. Modify your robots and programming in any way you want. The only rule is that any parts used by one robot have to also be available to all the other robots. For example, if one robot is using four

distance sensors, make sure you have enough distance sensors so that all the teams can have at least four, if they want. Parts should not give one team an advantage.

3. Once all robots are rebuilt, reprogrammed, and tested, have a maze-running competition. If you have time, allow each robot multiple chances, and only count each robot's best run.
4. OPTIONAL: If you have time, build a new maze, and without changing the robots or their programming, have another maze-running competition. Take each robot's best time, add it to the best time from the first maze, and the robot with the lowest overall combined time will be crowned the maze-running champion!

Post-Activity Check-Up

1. What was the algorithm you used to develop your program? Use a separate sheet of paper if there is not enough room here.

2. What Python functions and commands did you use to write your program? Briefly explain how you used each of them.

3. If you had more time, what would you change in your program to make it work better? How would you change your robot?

4. Describe the sensors you used in your robot, how you used them, and tell how well you think they worked.

5. What was the hardest part of this activity?

6. What advice would you give to students who are starting to build and program a maze-running robot?