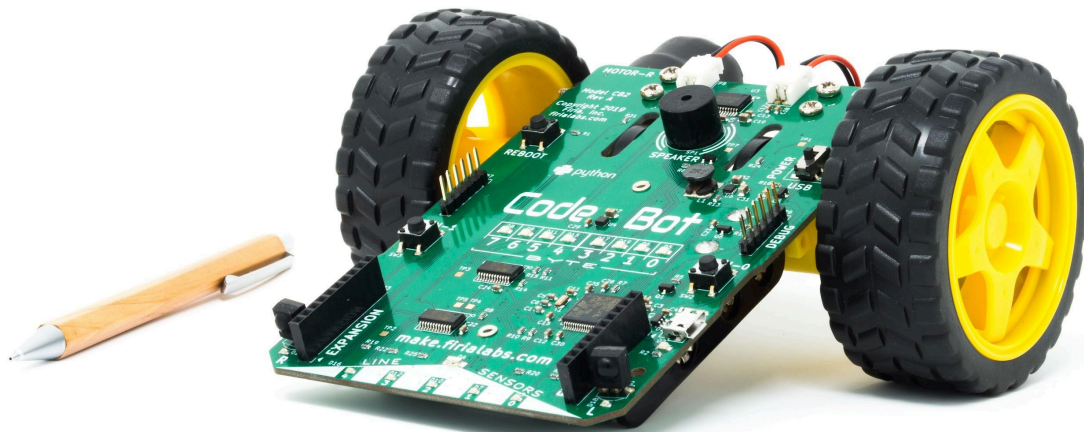


FIRIA LABS

Curriculum Guide



Mission Pack: Python with Robots

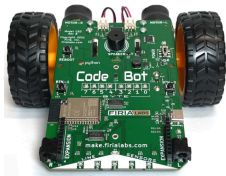


Table of Contents

Python with Robots Overview	2
Unit 1 Overview	3
Mission 1: Welcome	4
Mission 2: Introduction to CodeBot	5
Mission 3: Time and Motion	7
Unit 1 Remix Project and Exam	9
Unit 2 Overview	12
Mission 4: Animatronics	13
Mission 5: Fence Patrol	15
Unit 2 Remix Project and Exam	17
Unit 3 Overview	21
Mission 6: Line Follower	22
Mission 7: Hot Pursuit	24
Unit 3 Remix Project and Exam	26
Unit 4 Overview	30
Mission 8: Navigation	31
Mission 9: All Systems Go!	33
Unit 4 Remix Project and Exam	35
Final Project	39
Appendix A: Required Resources	44
Appendix B: Our Approach	45
Appendix C: Teacher Resources	46
Appendix D: Assessing Student Projects	48
Appendix E: Links to Teacher Materials	51
Appendix F: Lab Data Sheets	53



Python with Robots Overview



Designed as a Computer Science elective course for grades 8-12, this curriculum module covers the fundamentals of Python programming as students apply each new coding skill and concept to engaging projects with **CodeBot**. No prior coding experience is required! This 'bot puts the focus on coding, with built-in sensors and programmable controls for *endless* projects and learning opportunities.

Pre-Mission Assignment (5-10 hours)

If your students come with no Computer Science background, it is important to start by building a foundation of computational thinking. Dedicate some time for students to learn basic terms, such as algorithm, program, and debug. See the Firia Labs collection of Unplugged Activities at <https://learn.firia.com/curricula/cs-unplugged>.

Mission 1: Welcome



Take a tour of the CodeSpace Development Environment

Mission 2: Introducing CodeBot



Get to know your friendly neighborhood CodeBot!

Mission 3: Time and Motion



Power up the CodeBot. Get it moving in a square.

Mission 4: Animatronics



Create an “Animatronic Robot Exhibition” by utilizing the ‘bot’s speakers.

Mission 5: Fence Patrol



Stay between the lines to gain an in-depth understanding of CodeBot’s line sensors.

Mission 6: Line Follower



Tune up your Line Sensors and hit the road on the biggest and baddest line-course around. Can your Python code master this challenge?

Mission 7: Hot Pursuit



Go in-depth with the proximity sensors and write code to detect, pursue, and avoid objects.

Mission 8: Navigation



Learn to navigate by moving a specific direction, distance, and speed from a known location using the CodeBot’s wheel encoders.

Mission 9: All Systems Go!



Explore CodeBot’s internal sensor systems by creating a battery tester, temperature measurement tool, and alarm bot!



Unit 1: Getting Started (7-14 hours)

Students will learn about the programming environment, the CodeBot, and basic commands for programming the CodeBot using Python. Students create their own program to move the ‘bot in a simple shape, like a square and use button presses for input.

Summary of Mission 1:

Students start by completing the attitude survey. They create an account and join the class to access the curriculum. The mission will let them become familiar with CodeSpace.

Summary of Mission 2:

Then they learn about CodeBot, its peripherals, and proper care. Basic code, like importing a module and turning on an LED is introduced.

Summary of Mission 3:

Students learn several concepts and skills during this mission. First, the instructions go into depth about the LEDs and turning them on and off. There are three sets of LEDs on the ‘bot that can be controlled. Binary is introduced and how to use it in code to turn on/off LEDs. Students also learn about variables and how to use them in code. Finally, students learn how to turn on the motors and move the wheels both forwards and backwards. The mission ends with conditions and determining if a button was pressed. Since this mission covers many terms, concepts and coding statements, three review Kahoots were created.

Preparation and Materials:

- Create a class on the teacher dashboard.
- Students need a computer / laptop with the Chrome web browser.
- Make sure the students can successfully login to <http://make.firialabs.com>,
- Students create a student account and join the class with the code.
- Each student (or pair) needs a CodeBot and connecting cable.
- A ruler for measuring the distance traveled by the ‘bot.

Assessment:

Mission 2 Review Kahoot	Mission 3 Obj. 1-6 Kahoot	Mission 3 Obj 7-9 Kahoot	Mission 3 Obj 10-11 Kahoot
U1 Vocab Review Kahoot	U1 Coding Review Kahoot	U1 Vocab Test (MS Form)	U1 Coding Test (MS Form)

Standards addressed in this unit:

CSTA Standards Grades 6-8	CSTA Standards Grades 9-10	CSTA Standards Grades 11-12
<ul style="list-style-type: none"> ● 2-CS-03 ● 2-AP-10 ● 2-AP-11 ● 2-AP-13 ● 2-AP-19 	<ul style="list-style-type: none"> ● 3A-CS-03 ● 3A-AP-13 ● 3A-AP-16 ● 3A-AP-19 ● 3A-AP-21 	<ul style="list-style-type: none"> ● 3B-AP-17



Mission 1: Welcome	Time Frame: 1 hour
<p>Project Goal: Students will learn about the CodeSpace learning environment.</p> <p>Learning Targets</p> <ul style="list-style-type: none"> • I can navigate CodeSpace. • Identify major parts of the Codespace interface: Mission Bar, Objective Panel, text editor, CodeTrek, Toolbox, and Lesson Navigation Controls 	<p>Key Concepts</p> <ul style="list-style-type: none"> • Follow instructions in the Lesson Panel carefully. There is a lot of important reading! • Look for “tool icons” to collect tools in your Toolbox as you go.
<p>Assessment Opportunities</p> <ul style="list-style-type: none"> • Checkpoint 1.3 (toolbox) can be used as an exit ticket. • Quiz after Objective 4. • Print a picture of CodeSpace and have students label the parts. 	<p>Success Criteria</p> <ul style="list-style-type: none"> <input type="checkbox"/> Navigate CodeSpace <input type="checkbox"/> Identify major features of the CodeSpace interface: Editor panel, Lesson panel, Toolbox, CodeTrek, Hints
<p>Vocabulary</p> <ul style="list-style-type: none"> • Browser: Software that displays web pages • Cloud: A place to save files and data through the Internet • Objective: The steps in the mission; has a goal to accomplish • Text editor: Where you type the code • Code: Instructions to the computer • Toolbox: A place in CodeSpace to keep information you learn about programming concepts so you can use it later when you need the information • Simulation: A 3D environment that lets you see the robot move and interact in a virtual world 	
<p>New Python Code</p>	
<p>Real World Applications</p> <p>Programmers need to use some type of text editor to create their code. CodeSpace is an IDE, or integrated development environment. It is patterned after other popular IDEs.</p>	
<p>Teacher Notes:</p> <ul style="list-style-type: none"> • This lesson is the first lesson in all the mission packs. If your students have completed other mission packs with other physical devices, they will already know the information. You can choose to have them complete the mission as a review and refresher, or you can unlock the next mission. 	<p>Extensions / Cross-Curricular</p>



Mission 2: Introducing CodeBot		Time Frame: 1-2 hours													
<p>Project Goal: Students will learn about the peripherals of CodeBot and the basics of Python.</p> <p>Learning Targets</p> <ul style="list-style-type: none"> I can identify the main components of the CodeBot. I can safely connect and disconnect the CodeBot using the USB cable. I can create a new file. I can write code using the conventions of comments and correct punctuation. 		<p>Key Concepts</p> <ul style="list-style-type: none"> There are a lot of hardware peripherals on the CodeBot, including sensors, LEDs, motors, buttons, and a speaker. Python requires all objects – variables, peripherals, etc. – to be spelled exactly the same; capitalization matters! Adding comments and blank lines in your code makes it easier to read. 													
<p>Assessment Opportunities</p> <ul style="list-style-type: none"> Print a picture of the CodeBot and have students label the parts. Quiz after Objective 5. Exit ticket: What is the first index? Submit program after Objective 10. Mission 2 Review Kahoot 		<p>Success Criteria</p> <ul style="list-style-type: none"> <input type="checkbox"/> Identify the parts of the CodeBot <input type="checkbox"/> Create a new file <input type="checkbox"/> Import the botcore library and turn on an LED <input type="checkbox"/> Use descriptive comments 													
<p>Vocabulary</p> <ul style="list-style-type: none"> CodeBot: A computer on wheels with lots of sensors and controls built-in Peripherals: Devices that give input or output to the CodeBot; they include LED lights, speaker, motors, line sensors, proximity sensors, an accelerometer and push buttons Motors: Programmable electric engines; powers the wheels LEDs: Light emitting diodes; tiny and efficient electronic components that produce light Wheel encoders: Discs that rotate, counting the invisible IR light beam pulses through its slots Static electricity: A charge that can build up and causes a jolt and spark when grounded Comment: Notes in a program code that don't get executed (more information in Mission 3) Import: Provides access to a module, or library, of built-in Python functions to use in your code 															
<p>New Python Code</p> <table border="1" style="width: 100%;"> <tr> <td style="background-color: #FFF9C4;"><code>from botcore import leds</code></td> <td>Import from botcore only leds functions</td> </tr> <tr> <td style="background-color: #FFF9C4;"><code>leds.user_num(0, True)</code></td> <td>Turn on one user LED (parameters are LED number 0-7, True=on, False=off)</td> </tr> <tr> <td style="background-color: #FFF9C4;"><code>leds.ls_num(0, True)</code></td> <td>Turn on a line sensor LED (parameters are LED number 0-4, True/False)</td> </tr> </table>				<code>from botcore import leds</code>	Import from botcore only leds functions	<code>leds.user_num(0, True)</code>	Turn on one user LED (parameters are LED number 0-7, True=on, False=off)	<code>leds.ls_num(0, True)</code>	Turn on a line sensor LED (parameters are LED number 0-4, True/False)						
<code>from botcore import leds</code>	Import from botcore only leds functions														
<code>leds.user_num(0, True)</code>	Turn on one user LED (parameters are LED number 0-7, True=on, False=off)														
<code>leds.ls_num(0, True)</code>	Turn on a line sensor LED (parameters are LED number 0-4, True/False)														
<p>Real World Applications</p> <p>Make sure each student takes the time to personally inspect their 'bot. Discuss the fact that all the electronic devices they use have similar circuit boards inside. The tools and techniques they're learning apply to all the electronic devices they use every day! Challenge students to name a few devices they use every day that might contain computer chips or "microcontrollers" such as the one on the 'bot. How many of the following do they think of? There are so many more!</p> <table border="1" style="width: 100%; text-align: center;"> <tr> <td>Microwave oven</td> <td>Cell phone</td> <td>Automobile</td> <td>Watch or fitness tracker</td> </tr> <tr> <td>Video game controller</td> <td>Refrigerator</td> <td>Home thermostat</td> <td>Coffee maker</td> </tr> <tr> <td>Bread machine</td> <td>Alarm system</td> <td>Automatic garage doors</td> <td>Electronic locks</td> </tr> </table> <p>Challenge students to describe how our lives are impacted by the above technology, and to compare how related tasks were done before computer technology was invented.</p>				Microwave oven	Cell phone	Automobile	Watch or fitness tracker	Video game controller	Refrigerator	Home thermostat	Coffee maker	Bread machine	Alarm system	Automatic garage doors	Electronic locks
Microwave oven	Cell phone	Automobile	Watch or fitness tracker												
Video game controller	Refrigerator	Home thermostat	Coffee maker												
Bread machine	Alarm system	Automatic garage doors	Electronic locks												



Teacher Notes

- Review “input” and “output”. For each peripheral, discuss if it is used for input or output.
- [Lab sheets](#) and a [flowchart](#) are available.
- Discuss common programming practices, such as descriptive variable names, adding comments and blank lines, use of capital letters, etc.
- Discuss real-world applications, either at the beginning of the lesson or at the end of the lesson.

Extensions / Cross-Curricular

- Make a list of common input and output devices.
- **LANGUAGE ARTS:** Students write about technology today and its impact.
- **SCIENCE:** Students research a microcontroller or other every day device.
- **SCIENCE:** Have a lesson on LEDs and light.



Mission 3: Time and Motion	Time Frame: 2-6 hours
<p>Project Goal: Students will learn the basics of Python</p> <p>Learning Targets</p> <ul style="list-style-type: none"> • I can use the “Step” feature to debug a program. • I can use binary values to animate the LEDs. • I can use comments to explain my code. • I can assign data to a variable. • I can use variables to make code more efficient. • I can write an if:elif:else conditional statement. • I can use a button to control the ‘bot. 	<p>Key Concepts</p> <ul style="list-style-type: none"> • Computers execute code in sequential steps. • The CodeSpace debugger lets you <i>step</i> through the code one line at a time to understand what the computer is doing. • Built-in functions come from libraries, like botcore or time. • Variables can be defined to hold changing values. • Branching with if:elif:else statements controls the flow of the program. • The colon (:) at the end of an if statement introduces a new block of code. Everything inside the block should be indented at the same level.
<p>Assessment Opportunities</p> <ul style="list-style-type: none"> • Quiz after Objective 6 • Submit the “SequenceLEDs” program • Create a binary numbers quiz (5-digit & 8-digit) • Submit the “Binary LEDs” program • Submit the “MoveOut” program • Flowchart or pseudocode for square • Obj. 1-6 review Kahoot • Obj 7-9 review Kahoot • Obj 10-11 review Kahoot • Submit final “NavSquare” program <p>Supplemental Lab Data Sheets (see appendix)</p> <ul style="list-style-type: none"> • Obj 7 Get Moving and Obj 8 Rotation Time 	<p>Success Criteria</p> <ul style="list-style-type: none"> <input type="checkbox"/> Learn how to use the CodeSpace debugger <input type="checkbox"/> Flash CodeBot’s LEDs in a controlled sequence <input type="checkbox"/> Use the motors to move and rotate the ‘bot <input type="checkbox"/> Plan a program using a flowchart or pseudocode <input type="checkbox"/> Incrementally test code <input type="checkbox"/> Write code to drive in a specified pattern <input type="checkbox"/> Use if:elif:else conditional statements to perform actions based on conditions
<p>Vocabulary</p> <ul style="list-style-type: none"> • Physical computing: Writing code (instructions) for a physical device, like CodeBot or cars • Editor shortcuts: Keyboard hotkeys to write code faster; combinations of keys which complete a task • CPU: The “brain” of the computer that executes your code; the Central Processing Unit • Debugging: The process of understanding what the computer is actually doing and then changing the code to do what you want it to do • Delay: Functions that slow things down, like sleep(); the module must be imported first • Blocking functions: Functions that pause program execution; no other code will run during the pause • Literal: An actual value, like 1 or “hello” or True • Variable: A name to which you assign some data, any type of information your program uses; must be defined before it is used • Boolean: A value that is either True or False • Argument: Passing data to a function, determined by the position in the list when the function is called; arguments can be literal values, like True, or variables, like delay • Binary: How a computer deals with digits; electrical connections, like switches, that are either on or off • Byte: 8 bits of binary data • Comments: Notes in the code about what you are doing; increases the readability of code and is meant for humans, not the computer (they are not instructions to the computer and are not executed) • Whitespace: Adding blank lines and space around symbols to make the code more readable (ignored by Python, non-executable) • Algorithm: A precise sequence of instructions that the computer can follow exactly, one step at a time, to complete a task or solve a problem • Control flow branching: Decision points in code; code will take a different branch or path depending on a condition 	



- **Condition:** A Boolean value (True or False), often the result of a comparison operator like `<`, `>`, or `==`. Use an if statement, optionally followed by an elif or else, for branching
- **Indenting:** A way to structure blocks of code by offsetting a block of code four spaces; blocks of code are indented following a defining statement with a colon (:)

New Python Code

<code>from time import sleep</code>	Import the ability to delay
<code>sleep(1.0)</code>	Use <code>sleep()</code> – will sleep the amount of time in seconds
<code>delay = 1.0</code>	Define a variable (define variables near the top just under the imports)
<code>sleep(delay)</code>	Use a variable with <code>sleep()</code>
<code>leds.user_num(2, False)</code>	Turn off an LED
<code>leds.user(0b10101010)</code>	Use binary designation for turning on LEDs (0b for binary, then 0=off, 1=on for each LED)
<code>from botcore import *</code>	Import an entire library (* is a wildcard, which means everything)
<code>motors.enable(True)</code>	Turn on motors, must be done before motors will turn and wheels move
<code>motors.run(LEFT, 50)</code>	Turn left wheel forward at 50% power (use -50 for backward movement)
<code>motors.enable(False)</code>	Turn off motors
<code>buttons.was_pressed(0)</code>	Returns Boolean value – True if pressed, False if not pressed
<code>if buttons.was_pressed(0): elif buttons.was_pressed(1):</code>	Use button press in branching

Real World Applications

You've used some fundamental computer science and robotics principles:

- Controlling LEDs and motors with specific timing and sequencing
- Reading button inputs

This code is used in cars, stage lights, espresso machines, music sequencers, electric toothbrushes, and more!

Teacher Notes

- When you get to algorithms, you will need to stop and plan out the program with [flowcharts](#) or pseudocode. When testing, have tools, like tape and ruler, ready to go.
- As students test their code, have them change only ONE variable at a time. They can use the **Lab Data Sheet** to record their findings.
- As the robot performs the square, discuss factors that could affect performance, like battery power, type of surface, etc.
- There is a lot to the NavSquare program. You could potentially spread this out over several days in order to cover all the concepts embedded in this simple program.


Extensions / Cross-Curricular

- Instead of clockwise and counterclockwise squares, move the 'bot in a large square and a small square
- Move the 'bot in a square and a diamond
- Move the 'bot in a square and a rectangle
- **CHALLENGE:** Move the robot in a circle
- **MATH:** Use the 'bot and your lab data sheet to predict where the robot will go, given specific instructions. Then test it out.
- **MATH:** Draw a graph of the data from the lab data sheet.



Unit 1 Remix Project and Exam	Time Frame: 2-5 hours
<p>Remix Project Goal: Students will use the skills and concepts they learned in the first three missions to create their own project.</p> <p>Remix Project Outline: Follow the five-steps of the design process (document on next page) to design a remix project.</p>	<p>Remix Project Assessment Opportunities</p> <ul style="list-style-type: none"> • Peer reviews / Gallery walk • Remix 1 Log Planning Guide • Remix Rubric Checklist • Submit Remix Program <p>Unit 1 Exam Opportunities</p> <ul style="list-style-type: none"> • Unit 1 Vocabulary review Kahoot • Unit 1 Concepts and coding review Kahoot • Unit 1 Vocabulary test (MS Form) • Unit 1 Concepts & coding test (MS Form)
<p>Remix Project Ideas:</p> <ul style="list-style-type: none"> • Pick an extension idea from Mission 3. • Use the binary designation for turning on LEDs, and combine that with 'bot movement. • Light up different LEDs as the 'bot moves, to indicate what the movement is. • Program the 'bot like a remote control – when one button is pressed, it moves one way, and a different way for the other button. • Think of your own creative project with movement and LEDs and button input. 	
<p>Remix Rubric Checklist:</p> <ul style="list-style-type: none"> <input type="checkbox"/> Filename is descriptive <input type="checkbox"/> Uses one or more variables, each with a descriptive name <input type="checkbox"/> Moves the CodeBot forward and/or backward one or more times <input type="checkbox"/> Turns the CodeBot one or more times <input type="checkbox"/> Stops the CodeBot <input type="checkbox"/> Uses a sleep delay one or more times <input type="checkbox"/> Turns on one or more LED lights <input type="checkbox"/> Uses one or two buttons as input <input type="checkbox"/> Includes comments and whitespace for readability <input type="checkbox"/> Code follows programming conventions of indenting, punctuation and capitalization <input type="checkbox"/> Code runs with no errors 	



Unit 1 Remix Log	Name:	
Remix Step 1: Review your code from Mission 3		
Mission 3: Time and Motion What does this program do?		
What programming concepts did you learn and use?		
Remix Step 2: Remix Project Concept		
Look over the remix suggestions. Discuss with a partner. Then decide what you want to do for your remix project. Describe what your remix project will do:		
Remix Step 3: Plan your code. What variables will you use in the project? Fill out the charts below. Use another piece of paper to design your program with a flowchart or pseudocode.		
What variables will you use in the project? Fill in the chart. You do not need to fill in every line, or you can add more.		
	Variable Name	What it will be used for:
What buttons will you use, and what will happen when pressed?		
	Button	What will happen:
Remix Step 4: Write your code		
Use the sandbox  when you write the code. Write just a few lines at a time and test often.		
Remix Step 5: Commenting and feedback		
Documentation	<ul style="list-style-type: none"> • Make sure your code is readable by adding blank lines • Add comments to explain sections of code 	



Peer feedback: Get feedback from two (or more) people. You can be one of the peer reviewers.

Peer Review #1 Name:

Go through the checklist. Are all requirements met? If not, list any missing criteria.

What do you like about the program – be specific!

Give at least one suggestion. Begin with “what if” or “maybe you could”

Peer Review #2 Name:

Go through the checklist. Are all requirements met? If not, list any missing criteria

What do you like about the program – be specific!

Give at least one suggestion. Begin with “what if” or “maybe you could”

Review the comments. Then take time to improve or add to your project.

Post-Mission Reflection

What did you change in your project after reading the feedback?

What did you learn about programming from completing this project?

Rubric Checklist:

- Filename is descriptive
- Uses one or more variables, each with a descriptive name
- Moves the CodeBot forward and/or backward one or more times
- Turns the CodeBot one or more times
- Stops the CodeBot
- Uses a sleep delay one or more times
- Turns on one or more LED lights
- Uses one or two buttons as input
- Includes comments and whitespace for readability
- Code follows programming conventions of indenting, punctuation and capitalization
- Code runs with no errors



Unit 2: Inputs and Outputs (10-18 hours)

Students continue their programming journey by combining LED lights, movement and sound. They will learn about loops and creating and calling functions. CodeBot will interact with its environment using line sensors.

Summary of Mission 4:

Students are given the assignment to create an animatronic robot exhibit for a major theme park. It has some specific requirements. The first concept is an infinite loop used to flash the user lights in a cool pattern. Students learn about updating a variable and breaking out of a loop. Then they learn about using the speaker for making beeps and using random numbers. During this mission students also learn about while loops with a counter and use a button press for counting.

Summary of Mission 5:

Students learn about and use line sensors in code. Since CodeBot has 5 sensors, functions and loops are used to simplify and reuse code to access all line sensors and their LEDs. The code uses Boolean conditions and variables, and return functions. By the end of the mission, movement is added to line sensor input to keep the 'bot inside the lines of a fence.

Preparation and Materials:

- Students need a computer / laptop with the Chrome web browser.
- Make sure the students can successfully login to <http://make.firialabs.com>
- Students login with their account (possibly using their gmail account)
- Each student (or pair) needs a CodeBot and connecting cable.
- A metric ruler for measuring the distance traveled by the 'bot.
- Tape for a boundary that the 'bot stays within (needs to be the opposite color of the floor).

Assessment:

Mission 4 Obj. 1-5 Review Kahoot	Mission 4 Obj. 6-12 Review Kahoot	Mission 5 Review Kahoot
Unit 2 Vocab Review Kahoot	Unit 2 Coding Review Kahoot	U2 Vocab Test / Coding Test (MS Forms)

Standards addressed in this unit:

CSTA Standards Grades 6-8	CSTA Standards Grades 9-10	CSTA Standards Grades 11-12
<ul style="list-style-type: none"> • 2-CS-02 • 2-CS-03 • 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-13 • 2-AP-14 • 2-AP-15 • 2-AP-16 • 2-AP-17 • 2-AP-19 	<ul style="list-style-type: none"> • 3A-CS-02 • 3A-CS-03 • 3A-AP-13 • 3A-AP-16 • 3A-AP-17 • 3A-AP-18 • 3A-AP-19 • 3A-AP-21 • 3A-AP-22 	<ul style="list-style-type: none"> • 3B-CS-02 • 3B-AP-14 • 3B-AP-16 • 3B-AP-17 • 3B-AP-22 • 3B-AP-23



Mission 4: Animatronics	Time Frame: 4-6 hours								
<p>Project Goal: Students will create an Animatronic robot exhibit based on customer requirements.</p> <p>Learning Targets</p> <ul style="list-style-type: none"> • I can plan a project using a flowchart or pseudocode. • I can use a while True: loop. • I can increment a counter variable. • I can use buttons.was_pressed to control a variable. • I can write a function. • I can import the random library and use it to generate random numbers. 	<p>Key Concepts</p> <ul style="list-style-type: none"> • While loops are used to execute an algorithm constantly. • Increments (and decrements) are used for updating variables like counters. • Button presses (inputs), LEDs (outputs) and speaker sounds (outputs) are part of the user interface. They allow the user to interact with the CodeBot. • Python’s random library makes it easy to work with random numbers. • A function is a named chunk of code you can run anytime by calling its name. 								
<p>Assessment Opportunities</p> <ul style="list-style-type: none"> • Quiz after Objective 7 • Submit flowchart or pseudocode for the project • Map out a fanfare with different notes • Give a code review worksheet with code segments that include incrementing and loops. Have students trace the code. • Obj. 1-5 review Kahoot • Obj 6-12 review Kahoot • Submit final “SweepLEDs” program 	<p>Success Criteria</p> <ul style="list-style-type: none"> <input type="checkbox"/> Create a flowchart or pseudocode that includes all elements from the napkin sketch. <input type="checkbox"/> Use loops correctly to blink LEDs until a break. <input type="checkbox"/> Use an increment to light LEDs in sequence. <input type="checkbox"/> Use an increment to count button presses. <input type="checkbox"/> Use randrange to play a random pitch. <input type="checkbox"/> Define a function to play a single note. <input type="checkbox"/> Execute all elements from the napkin sketch. 								
<p>Vocabulary</p> <ul style="list-style-type: none"> • Loop: Changing the flow of the code by repeating a block of code, subject to a condition. • While condition: A statement that tells Python to repeat the block of code as long as the given condition is true. • Infinite loop: A loop that never ends because the condition is always true. • Updating a variable: Assign a new value to a variable, based on the old value. • Increment: Update a variable by adding one (or a specific number) to the old value. • Single equal (=): Assignment symbol – used to assign a value to a variable. • Double equal (==): Comparison operator to determine if two objects are the same. • Break: Exit the nearest enclosing loop. • Debounce: Reset the internal status of a button so the press isn’t counted twice. • While loop: A loop that iterates, or repeats, while a condition is true. • Parameter: A list of names declared in a function definition that receive values when the function is called and act like local variables in the function. • Random number: When using randrange, a range of numbers is given (start, stop). The random number will include the start but will be less than the stop. • Function: A named chunk of code you can run anytime just by calling its name; reuse code without retyping it. 									
<p>New Python Code</p> <table border="1"> <tr> <td><code>while True:</code></td> <td>Infinite loop</td> </tr> <tr> <td><code>n_led = n_led + 1</code></td> <td>Incrementing or updating a variable</td> </tr> <tr> <td><code>n_led = 0</code></td> <td>Defining or resetting a variable</td> </tr> <tr> <td><code>break</code></td> <td>Break out of the nearest loop</td> </tr> </table>		<code>while True:</code>	Infinite loop	<code>n_led = n_led + 1</code>	Incrementing or updating a variable	<code>n_led = 0</code>	Defining or resetting a variable	<code>break</code>	Break out of the nearest loop
<code>while True:</code>	Infinite loop								
<code>n_led = n_led + 1</code>	Incrementing or updating a variable								
<code>n_led = 0</code>	Defining or resetting a variable								
<code>break</code>	Break out of the nearest loop								



<code>n_guests = n_guests + 1</code>	Increment (also count = count + 1)
<code>leds.ls_num(n_guests, True)</code>	Turn on LED using a variable
<code>spkr.pitch(440) / sleep(0.1)</code>	Play a tone on the speaker
<code>spkr.off()</code>	Turn off the speaker
<code>buttons.was_pressed(0)</code>	Used by itself to debounce a button press
<code>while count < 10:</code>	While loop (will iterate or repeat 10 times if count starts at 0)
<code>from random import randrange</code>	Import random library
<code>f = randrange(100, 1000)</code>	Get a random number within a range
<code>def flashLEDs():</code> <code>def note(freq, duration):</code>	Define a function Parameters are included in the ()
<code>flashLEDs() / note(F4, 0.4)</code>	Call a function (arguments are included in the parenthesis)

Real World Applications

- Movies, art, theme parks, and more use code to create cool special effects!
- Traffic monitors use pressure switches to count traffic.
- Traffic lights sometimes have inductive coils on the surface to trigger the light to change.

Teacher Notes

- At the beginning of the lesson, require students to create a flowchart or pseudocode and go over the engineering design process.
- Trace the variables in the code together. Use the debugger, use a display screen, or make a large chart. Note that the loop doesn't stop at 7, there just aren't any more LEDs left.
- Review the difference between = and ==.
- Students can refer to their Lab Data Sheet from Mission 3 to help with moving forward and turning.

Extensions / Cross-Curricular

- Make the animation longer by having the CodeBot do more things.
- Wait for more than 5 people before triggering the animation. Use a random number!
- Add code so when a button is pressed, the CodeBot returns to its original position.
- **LANGUAGE ARTS:** Have students write a story about a robot at a theme park.
- **PERFORMING ARTS:** Learn about music and note notation.



Mission 5: Fence Patrol	Time Frame: 4-6 hours								
<p>Project Goal: Students will gain an in-depth understanding of CodeBot's line sensors.</p> <p>Learning Targets</p> <ul style="list-style-type: none"> I can print values in the console panel while debugging to get real-time sensor values. I can use ls.read() to get real-time line sensor values. I can make a contact counter to show each line-detect on the user LEDs. I can teach the 'bot to stay inside the lines. I can define a function that returns a Boolean value. 	<p>Key Concepts</p> <ul style="list-style-type: none"> Analog sensors are non-contact sensors used in many industrial and commercial applications. Use threshold comparisons to make decisions with sensor data. The console panel can be used to print real-time data using the print() statement. Engineers build in safety features, like waiting for a button press before starting. Autonomous robots use sensor data to make decisions and take action in its unique environment. A dark line on a light background will have a smaller value; a light line on a dark background will have a larger value. 								
<p>Assessment Opportunities</p> <ul style="list-style-type: none"> Quiz after Objective 5 Submit flowchart or pseudocode for the project Submit the data record sheet of objects and their sensor readings Use the lab sheets for the mission. Explain how a counter variable works Review Kahoot Submit final "LineSense" program <p>Supplemental Lab Data Sheets (see appendix)</p> <ul style="list-style-type: none"> Obj 2 Line Sensors Obj 2 The Debug Console 	<p>Success Criteria</p> <ul style="list-style-type: none"> <input type="checkbox"/> Read data from a line sensor and display it on the Console Panel. <input type="checkbox"/> Use a variable for the threshold that is specific for the testing environment. <input type="checkbox"/> Use a condition with the threshold to determine if a line is detected. <input type="checkbox"/> Define a function for checking one sensor. <input type="checkbox"/> Define a function for checking all line sensors. <input type="checkbox"/> Define a function that turns on the LED above a sensor if a line is detected. <input type="checkbox"/> Count each time a line is detected. <input type="checkbox"/> Reuse code from Mission 3 to drive the 'bot. 								
<p>Vocabulary</p> <ul style="list-style-type: none"> Line sensors: Photo reflective sensors that detect lines and boundaries beneath the CodeBot API: Application Programming Interface – the details of how your program interacts with different services it needs Analog: Infinite variation, like from dark to light or cold to hot ADC: Analog to digital converter REPL: "Read, evaluate, print loop" command line that enables print statement output DRY: Don't Repeat Yourself – never write the same code twice Return statement: Exits the function and sends a value back to the code where the function was called 									
<p>New Python Code</p> <table border="1"> <tbody> <tr> <td data-bbox="142 1539 727 1633"> <pre>ls.read(num) / val = ls.read(n)</pre> </td> <td data-bbox="727 1539 1469 1633"> Reads a line sensor. The sensor num can be 0-4, returns a value between 0-4095 </td> </tr> <tr> <td data-bbox="142 1633 727 1728"> <pre>print(val) print("line sensor value = ", val)</pre> </td> <td data-bbox="727 1633 1469 1728"> Display the value of a variable on the console </td> </tr> <tr> <td data-bbox="142 1728 727 1854"> <pre>threshold = 2500 is_detected = val < threshold leds.ls_num(0, is_detected)</pre> </td> <td data-bbox="727 1728 1469 1854"> Assign a Boolean result of a comparison to a variable. Use the Boolean variable in code. </td> </tr> <tr> <td data-bbox="142 1854 727 1948"> <pre><Line detection></pre> </td> <td data-bbox="727 1854 1469 1948"> Dark line on light surface – use val > threshold Light line on dark surface – use val < threshold </td> </tr> </tbody> </table>		<pre>ls.read(num) / val = ls.read(n)</pre>	Reads a line sensor. The sensor num can be 0-4, returns a value between 0-4095	<pre>print(val) print("line sensor value = ", val)</pre>	Display the value of a variable on the console	<pre>threshold = 2500 is_detected = val < threshold leds.ls_num(0, is_detected)</pre>	Assign a Boolean result of a comparison to a variable. Use the Boolean variable in code.	<pre><Line detection></pre>	Dark line on light surface – use val > threshold Light line on dark surface – use val < threshold
<pre>ls.read(num) / val = ls.read(n)</pre>	Reads a line sensor. The sensor num can be 0-4, returns a value between 0-4095								
<pre>print(val) print("line sensor value = ", val)</pre>	Display the value of a variable on the console								
<pre>threshold = 2500 is_detected = val < threshold leds.ls_num(0, is_detected)</pre>	Assign a Boolean result of a comparison to a variable. Use the Boolean variable in code.								
<pre><Line detection></pre>	Dark line on light surface – use val > threshold Light line on dark surface – use val < threshold								



<pre>n = 0 while n < 5: detect_line(n) n = n + 1</pre>	Use a comparison with a while loop and counter variable to repeat a specific number of times
<pre>while True: if buttons.was_pressed(): break</pre>	Wait loop for safe driving. The loop will wait and do nothing until the button is pressed.
<pre>return is_detected / return got_line</pre>	Return statement that gives a value back to the function call
<pre>hit = scan_lines() if detect_line(count)</pre>	Call a function that returns a value. It can be used as an assignment, or in a condition.
<pre>leds.user(line_count)</pre>	Use a variable to turn on LEDs. line_count will be from 0-255
<pre>line_count = line_count + 1 if line_count == 256: line_count = 0</pre>	Wrap-around the line_count variable for binary numbers. This code will “reset” a variable to its initial value once it exceeds its possible values.

Real World Applications

- Automatic Guided Vehicles (AGVs) use this kind of code to zoom around warehouse distribution centers, getting packages to you!
- Robots are used to clean up environmental waste, explore underground mines, discover shipwrecks, and do other tasks deemed unsafe for humans.

Teacher Notes

- Have students create flowchart sometime before defining the detect_line() and scan_lines() functions.
- Students should use the Console Panel to track variables as they work to write the functions.
- Use at least one of the Lab Data Sheets to practice reading the line sensor and different reflective materials. They can check out materials other than tape on the floor – let them have fun with this. What kind of reading does their notebook or backpack give? Does the lighting matter?
- You may want to review incrementing a variable and how a counter can be used to control a while loop for specific repeating.
- You may want to review using a variable for lighting LEDs.

Extension / Cross-Curricular

- Add a button for stopping the CodeBot.
- Use one button for dark on light readings, and the other button for light on dark readings.
- Use one button for a lot of light in the room, and the other button for a darker setting.
- Add sounds to the code so the ‘bot gives a specific auditory announcement when it hits a line.
- Use random numbers – it could be for speed, amount of turn, etc.
- **LANGUAGE ARTS:** Have students write a poem about programming or robots.
- **SCIENCE:** Explore light sensors and how they work.
- **SCIENCE:** Discuss the difference between analog and digital – give examples.
- **PHYSICAL SCIENCE:** Do experiments with different speeds and thresholds. Which ones work the best? When is the ‘bot not able to detect a line? Make predictions and then test them.
- **MATH:** Make a chart out of the data on the recording sheet.
- **ART:** Attach a marker to the ‘bot and have it draw a picture as it stays within the lines.




Unit 2 Remix Project and Exam	Time Frame: 2-5 hours
<p>Remix Project Goal: Students will use the skills and concepts they learned in Mission 4 and Mission 5 to create their own project.</p> <p>Remix Project Outline: Follow the five-steps of the design process (document on next page) to design a remix project.</p>	<p>Remix Project Assessment Opportunities</p> <ul style="list-style-type: none"> ● Peer reviews / Gallery walk ● Remix 2 Log planning guide ● Remix Rubric ● Submit Remix Program <p>Unit 2 Exam Opportunities</p> <ul style="list-style-type: none"> ● Unit 2 Vocabulary review Kahoot ● Unit 2 Concepts and coding review Kahoot ● Unit 2 Vocabulary test (MS Form) ● Unit 2 Concepts & coding test (MS Form)
<p>Remix Project Ideas:</p> <ul style="list-style-type: none"> ● Pick an extension idea from Mission 4 or Mission 5. ● Combine Mission 4 and Mission 5 using the buttons: press 0 for animatronics and 1 for fence patrol. ● Add a line detection function to the animatronics program so it doesn't "fall off the stage" ● Add sounds and actions to the fence patrol mission. ● Think of your own creative project with movement and LEDs and button input. 	
<p>Remix Rubric Checklist:</p> <ul style="list-style-type: none"> <input type="checkbox"/> Filename is descriptive <input type="checkbox"/> Uses one or more variables, each with a descriptive name <input type="checkbox"/> Moves the CodeBot forward and/or backward one or more times <input type="checkbox"/> Turns the CodeBot one or more times <input type="checkbox"/> Plays a sound <input type="checkbox"/> Turns on one or more LED lights <input type="checkbox"/> Uses one or two buttons as input <input type="checkbox"/> Uses the line sensors to control the CodeBot <input type="checkbox"/> Defines at least one function with a return <input type="checkbox"/> Code follows programming conventions of comments, readability, indenting, and capitalization <input type="checkbox"/> Code runs with no errors 	



Unit 2 Remix Log	Name: _____									
Remix Step 1: Review your code from Mission 4 and 5										
Mission 4: Animatronics What does this program do? Mission 5: Fence Patrol What does this program do?										
What programming concepts did you learn and use in each mission?										
Remix Step 2: Remix Project Concept										
Look over the remix suggestions. Discuss with a partner. Then decide what you want to do for your remix project. Describe what your remix project will do:										
Remix Step 3: Plan your code. What variables will you use in the project?										
Fill out the charts below. Use another piece of paper to design your program with a flowchart or pseudocode.										
What variables will you use in the project? Fill in the chart. You do not need to fill in every line, or you can add more.	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 40%;">Variable Name</th> <th>What it will be used for:</th> </tr> </thead> <tbody> <tr><td style="height: 20px;"></td><td></td></tr> <tr><td style="height: 20px;"></td><td></td></tr> <tr><td style="height: 20px;"></td><td></td></tr> </tbody> </table>		Variable Name	What it will be used for:						
Variable Name	What it will be used for:									
What buttons will you use, and what will happen when pressed?	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 25%;">Button</th> <th>What will happen:</th> </tr> </thead> <tbody> <tr><td style="height: 20px;"></td><td></td></tr> <tr><td style="height: 20px;"></td><td></td></tr> </tbody> </table>		Button	What will happen:						
Button	What will happen:									
What functions will you write? Describe each one.	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 40%;">Function name</th> <th>What it will do</th> </tr> </thead> <tbody> <tr><td style="height: 20px;"></td><td></td></tr> <tr><td style="height: 20px;"></td><td></td></tr> </tbody> </table>		Function name	What it will do						
Function name	What it will do									



Remix Step 4: Write your code

Use the sandbox  when you write the code. Write just a few lines at a time and test often.

Remix Step 5: Commenting and feedback

Documentation

- Make sure your code is readable by adding blank lines
- Add comments to explain sections of code

Peer feedback: Get feedback from two (or more) people. You can be one of the peer reviewers.

Peer Review #1 Name:

Go through the checklist. Are all requirements met? If not, list any missing criteria.

What do you like about the program – be specific!

Give at least one suggestion. Begin with “what if” or “maybe you could”

Peer Review #2 Name:

Go through the checklist. Are all requirements met? If not, list any missing criteria

What do you like about the program – be specific!

Give at least one suggestion. Begin with “what if” or “maybe you could”

Review the comments. Then take time to improve or add to your project.

Post-Mission Reflection

What did you change in your project after reading the feedback?

What did you learn about yourself from completing this project?



Unit 2 Remix Rubric Checklist:

- Filename is descriptive
- Uses one or more variables, each with a descriptive name
- Moves the CodeBot forward and/or backward one or more times
- Turns the CodeBot one or more times
- Plays a sound
- Turns on one or more LED lights
- Uses one or two buttons as input
- Uses the line sensors to control the CodeBot
- Defines at least one function with a return
- Code follows programming conventions of comments, readability, indenting, and capitalization
- Code runs with no errors



Unit 3: Get Moving (10-18 hours)

Students learn more about CodeBot and programming. They will use line sensors and proximity sensors to control the CodeBot. Students will utilize lists in their code to make it more efficient. They will use math to calibrate the sensors and inform the decision-making of the 'bot.

Summary of Mission 6:

The project in this mission is to have CodeBot follow a line. The line can be dark on light or light on dark. Students will learn to calibrate the line sensors and use code to precisely follow a line. They will also use sound as an indicator for completing a task.

Summary of Mission 7:

Students will use the built-in proximity sensors to detect objects. They will calibrate the 'bot so it can adapt to its environment, depending on the reflectivity of the surface. The final project will enable the 'bot to follow an object, like a curious puppy that chases a ball.

Preparation and Materials:

- Students need a computer / laptop with the Chrome web browser.
- Make sure the students can successfully login to <http://make.firialabs.com>
- Students login with their account (possibly using their gmail account)
- Each student (or pair) needs a CodeBot and connecting cable.
- Tape for a line that the 'bot can follow (needs to be the opposite color of the floor).
- Objects for the proximity sensors to detect in front of the 'bot.
- Metric ruler and different surfaces for testing

Assessment:

Mission 6 Review Kahoot	Mission 7 Review Kahoot	Unit 3 Vocabulary Test (Microsoft Form)
Unit 3 Vocab Review Kahoot	Unit 3 Coding Review Kahoot	Unit 3 Coding and Concept Test (MS Form)

Standards addressed in this unit:

CSTA Standards Grades 6-8	CSTA Standards Grades 9-10	CSTA Standards Grades 11-12
<ul style="list-style-type: none"> • 2-CS-01 • 2-CS-02 • 2-CS-03 • 2-DA-08 • 2-DA-09 • 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-13 • 2-AP-14 • 2-AP-15 • 2-AP-16 • 2-AP-17 • 2-AP-18 • 2-AP-19 	<ul style="list-style-type: none"> • 3A-CS-01 • 3A-CS-02 • 3A-CS-03 • 3A-AP-13 • 3A-AP-14 • 3A-AP-15 • 3A-AP-16 • 3A-AP-17 • 3A-AP-18 • 3A-AP-19 • 3A-AP-21 • 3A-AP-22 • 3A-AP-23 • 3A-AP-26 	<ul style="list-style-type: none"> • 3B-CS-02 • 3B-AP-10 • 3B-AP-12 • 3B-AP-14 • 3B-AP-15 • 3B-AP-16 • 3B-AP-17 • 3B-AP-21 • 3B-AP-22 • 3B-AP-23



<p>Mission 6: Line Follower</p>	<p>Time Frame: 4-6 hours</p>
<p>Project Goal: Students use sensor input to program the ‘bot to follow a line.</p> <p>Learning Targets</p> <ul style="list-style-type: none"> • I can use REPL to gather real-time data from sensors. • I can call the <code>is.check()</code> function to use the ADC hardware channel scanning feature. • I can use logic operators to increase the reliability and speed of the ‘bot. • I can use tuples to create proportional speed control. • I can use built-in math functions to calibrate the CodeBot. 	<p>Key Concepts</p> <ul style="list-style-type: none"> • There is no “hidden magic” going on here! The ‘bot responds to programmed inputs and outputs. • The line follower ‘bot will need to continuously check for the presence of a line beneath all five sensors. • A list and a tuple are similar, but different. A list is mutable, or can change, and tuple is immutable, or cannot be modified. • Pre-coded functions from the botcore library can use the ADC hardware, making your program even faster! • Auto-calibration algorithms can make your ‘bot adaptable to any environment. • Global variables exist outside a function, while local variables are created inside a function. You can declare a variable global in a function. • Python has many built-in functions for math.
<p>Assessment Opportunities</p> <ul style="list-style-type: none"> • Quiz after Objective 2 • Quiz after Objective 8 • Submit flowchart or pseudocode for the project • Submit the “CheckLines” program • Review Kahoot • Submit final “LineFollow1” program <p>Supplemental Lab Data Sheets (see appendix)</p> <ul style="list-style-type: none"> • Obj 2 Using REPL • Obj 3 Line Sensing • Obj 4 New Threshold • Obj 5 Between the Ed • Obj 5 Line Follower Reflection 	<p>Success Criteria</p> <ul style="list-style-type: none"> <input type="checkbox"/> Create a basic line follower program using two edge sensors <input type="checkbox"/> Improve the program with a center line sensor to keep the ‘bot straight <input type="checkbox"/> Use all five line sensors for proportional steering control <input type="checkbox"/> Adapt to your environment with line calibration code
<p>Vocabulary</p> <ul style="list-style-type: none"> • List: A sequence of items you can access with an index • Tuple: Read-only form of a list • Or (logical operator): Multiple conditions to compare, testing if any one (or both) is true • Hard coded values: Specific values used in code that can be replaced with a variable or constant • Globals: Variables defined outside a function in the main program; they are available and can be accessed during the entire program execution • Locals: Variables defined inside a function; they only exist during the function execution and can only be accessed in the function • Int (integer): A value that is an integer; designated as int in Python; can be positive or negative • Float (decimal): A value that is a decimal, also known as floating point; can be positive or negative. • Auto-calibrate: Use CodeBot sensors to automatically adapt to its environment by detecting lines and objects and setting parameters like <code>is_reflective</code> and <code>thresh</code>. 	
<p>New Python Code</p>	
<pre>detected = [False, False, False, False, False]</pre>	<p>Create a list</p>
<pre>detected[count] = val > thresh</pre>	<p>Update a specific value in a list</p>



<pre>leds.ls([False, False, False, False, False]) vals = check_lines(threshold) leds.ls(vals)</pre>	Use a list with LEDs
<pre>vals = ls.check(thresh, is_reflective) leds.ls(vals)</pre>	Botcore line sensors function (similar to check_lines but faster). It takes 2 parameters and returns a tuple.
<pre>elif vals[1] or vals[2] or vals[3]:</pre>	Using the or logical operator. Can have 2 or more conditions, any of which can be true and make the entire condition true
<pre>elif vals == (0, 1, 1, 0, 0):</pre>	Comparing with a tuple
<pre>global count / global thresh, is_reflective</pre>	Code needed to indicate a global variable inside a function
<pre>abs(x) / round(x, ndigits)</pre>	Built-in math operations (math library)

Real World Applications

Self-driving cars, autonomous flying drones and other computing systems that navigate on their own have some basic principles in common. Whether writing code for a vehicle with a high-powered vision processing system or for CodeBot's low-power sensors, you will face many of the same challenges to achieve the objective. Based on sensor inputs, what actions should you take to stay on the path? Robots that zip through warehouse distribution centers often follow lines as they pick up and pack items you order when shopping online.

Teacher Notes

- There is a lot going on during this mission! Take your time and don't rush through the objectives.
- Use REPL and the Lab Data Sheets to make sense of what the data returned by the sensors and what the 'bot is doing with it.
- Many Lab Data Sheets are available. Use the ones that are helpful to your students.
- Competition! Lay out a track and see which 'bot gets to the finish line first.

Extensions / Cross-Curricular

- Add a button to stop the 'bot.
- Add more sounds in the code. Play a sound when on the line, or a different sound when going off the line.
- Light up user LEDs as a communication on what the 'bot is doing.
- Have a competition on a track to see who's 'bot gets to the finish line first. If it is yours, play a victory song and do a dance!
- **LANGUAGE ARTS:** Have students compare and contrast how the line sensors are used in Mission 5 and Mission 6. What are the similarities and differences?
- **SCIENCE:** Discuss friction and turning speeds. Experiment with the 'bot on slow turning speeds vs fast turning speeds and what type of angles they work the best on. Try the 'bot on different surfaces.
- **MATH:** Make charts out of the data on the recording sheet.



<p>Mission 7: Hot Pursuit</p>	<p>Time Frame: 4-6 hours</p>
<p>Project Goal: Students will use proximity sensors to program the 'bot to track and chase an object.</p> <p>Learning Targets</p> <ul style="list-style-type: none"> • I can use the proximity sensor to detect objects. • I can experiment with light and dark surfaces to find the ideal power and threshold settings for each environment. • I can write calibration functions so the 'bot can adapt to its environment. • I can apply previous knowledge of the motors to rotate and face an object moving in front. • I can follow an algorithm to track an object and chase after it. 	<p>Key Concepts</p> <ul style="list-style-type: none"> • CodeBot uses the Infrared Proximity Sensor system to detect objects in its path. • A detection threshold of 0-100% controls how much light is needed for a True detection. If you decrease the thresh value, the 'bot works well even on a white surface. • An emitter power level setting from 1 to 8 (high power) controls the brightness of CodeBot's IR "flashlight." • The prox.detect(power, thresh) function lets you adapt to different environments. • Using auto calibration functions for power and thresh allows the 'bot to adapt to a new environment.
<p>Assessment Opportunities</p> <ul style="list-style-type: none"> • Quiz after Objective 5 • Have students explain the power and threshold used in prox.detect() • Have students explain how a variable can be used to "toggle" a Boolean value. • Review Kahoot • Submit final "HotPursuit" program <p>Supplemental Lab Data Sheets (see appendix)</p> <ul style="list-style-type: none"> • Obj 1 Presence Detector Experiment • Obj 2, 4, 7 Lab Data Sheet • Obj 1 Surface Test • Obj 2 Power / Obj 2 Threshold • Mission 7 Reflection 	<p>Success Criteria</p> <ul style="list-style-type: none"> <input type="checkbox"/> Use the basic proximity sensor prox.detect() to detect objects in front of the 'bot. <input type="checkbox"/> Use prox.range() to find the best threshold for detecting a reflection. <input type="checkbox"/> Write calibration functions for power and threshold. <input type="checkbox"/> Use motors to follow a detected object. <input type="checkbox"/> Calculate and use a turn ratio for a more smooth movement when chasing an object.
<p>Vocabulary</p> <ul style="list-style-type: none"> • Proximity sensors: Infrared (IR) sensors that can detect nearby objects based on reflected IR light • Detection sensitivity: How much light is needed for the proximity sensor to detect an object (0 to 100) • Emitter power level: The brightness of CodeBot's IR flashlight, with settings from 1 to 8 (high power) • not (logical operator): A special kind of logical operator that needs only one Boolean operand, and inverts it; it can be used to toggle a Boolean variable 	
<p>New Python Code</p>	
<pre>prox.detect()</pre>	<p>Read the proximity sensors; returns a tuple (left, right) with values True or False</p>
<pre>vals = prox.detect() left_detect = vals[0] right_detect = vals[1]</pre>	<p>Using the detect function – it returns a tuple Vals[0] could also be vals[LEFT] Vals[1] could also be vals[RIGHT]</p>
<pre>p=prox.detect() leds.prox(p)</pre>	<p>Use the True/False values of detect() to turn on/off the LEDs by the proximity sensors</p>
<pre>prox.detect(power, threshold)</pre>	<p>Use parameters with the function. Power is the "flashlight" with settings from 1 to 8 (high) Threshold is the sensitivity level, with settings from 0 to 100 (how much light is needed to detect an object)</p>



<pre>prox.range() prox.range(samples, power, low, high)</pre>	<p>A function that calculates the ideal threshold in an environment. It returns a tuple – two numbers, a threshold for left and a threshold for right. Parameters are optional.</p>
<pre>go_motors = False go_motors = not go_motors</pre>	<p>Use the logical operator not to “toggle” the go_motors Boolean value: will go from True to False to True, etc.</p>

Real World Applications

The kind of code you’ve written is inside electronic objects you might use every day, without even thinking about it! Some examples are given below. Can you think of even more?

- Touchless faucets
- soap dispensers and hand dryers
- automatic doors
- vehicle navigation and safety systems
- factory automation systems

Teacher Notes

- For testing surfaces, you can use cardstock, construction paper, etc of different colors. Ideally each student/pair should have at least white and black and a color in between. You could also use material, shiny surfaces, etc.
- Natural light will impact the results. If you have windows, morning and afternoon classes may get different numbers.
- The person measuring distance should not be in range of the sensors!
- One sensor may be more sensitive than the other. Decide what you will measure and be consistent!
- Several Lab Data Sheets are also available. Use the ones that are helpful to your students.
- The REPL console log streams at a very fast pace. Students can slow it down with a sleep() statement.
- Once again, there is a lot of information given in this lesson, and sometimes the steps go quickly. Take your time with each objective and review the concepts frequently.
- Give examples of things that need calibration, and how you might do the calibration. In this program, students start with one calibration function and then use it in another to do a second calibration. This can be tricky. You might want to work out an example on the board.

Extensions / Cross-Curricular

- Add sound to the chase.
- Add an animation. Keep track of the time. If the object hasn’t moved in 10 seconds, have the ‘bot perform an animation, like a puppy asking for the ball to be thrown.
- Add an animation for when there is no object detected, like a puppy wanting to play.
- **LANGUAGE ARTS:** Have students write a summary of their project, using technical terms.
- **SCIENCE:** Have a lesson on IR.
- **SCIENCE:** Turn some of the objectives into experiments. Make an hypothesis and go through the scientific method to prove or disprove.
- **MATH:** The last objective uses a turn ration. Have a lesson on ratios.
- **MATH:** The lesson gives the possibility of having a lot of test data recorded. Use the data to make inferences, charts, etc.



Unit 3 Remix Project and Exam	Time Frame: 2-5 hours
<p>Remix Project Goal: Students will use the skills and concepts they learned in Mission 6 and Mission 7 to create their own project.</p> <p>Remix Project Outline: Follow the five-steps of the design process (document on next page) to design a remix project.</p>	<p>Remix Assessment Opportunities</p> <ul style="list-style-type: none"> • Peer reviews / Gallery walk • Remix 3 Log planning guide • Remix Rubric Checklist • Submit Remix Program <p>Unit 3 Exam Opportunities</p> <ul style="list-style-type: none"> • Unit 3 Vocabulary review Kahoot • Unit 3 Concepts and coding review Kahoot • Unit 3 Vocabulary test (MS Form) • Unit 3 Concepts & coding test (MS Form)
<p>Remix Project Ideas:</p> <ul style="list-style-type: none"> • Pick an extension idea from Mission 6 or Mission 7. • Start with a line follower program and then allow the ‘bot to be distracted by a detected object and go “off course.” • Add sounds and animations to one of the mission projects. • Think of your own creative project with line or proximity sensors, movement, LEDs and button input. 	
<p>Remix Rubric Checklist:</p> <ul style="list-style-type: none"> <input type="checkbox"/> Filename is descriptive <input type="checkbox"/> Uses one or more variables, each with a descriptive name <input type="checkbox"/> Moves the CodeBot forward and/or backward one or more times <input type="checkbox"/> Turns the CodeBot one or more times <input type="checkbox"/> Turns on one or more LED lights <input type="checkbox"/> Uses one or two buttons as input <input type="checkbox"/> Uses at least one sensor to control the CodeBot <input type="checkbox"/> Defines at least one function with a return <input type="checkbox"/> Includes something extra (sound, more than one sensor, more than one function, etc.) <input type="checkbox"/> Code follows programming conventions of comments, readability, indenting, and capitalization <input type="checkbox"/> Code runs with no errors 	




Unit 3 Remix Log	Name:	
Remix Step 1: Review your code from Mission 6 and 7		
Mission 4: Line Follower What does this program do?		
Mission 5: Hot Pursuit What does this program do?		
What programming concepts did you learn and use in each mission?		
Remix Step 2: Remix Project Concept		
Look over the remix suggestions. Discuss with a partner. Then decide what you want to do for your remix project. Describe what your remix project will do:		
Remix Step 3: Plan your code. What variables will you use in the project? Fill out the charts below. Use another piece of paper to design your program with a flowchart or pseudocode.		
What variables will you use in the project? Fill in the chart. You do not need to fill in every line, or you can add more.		
	Variable Name	What it will be used for:
What functions will you write? Describe each one.		
	Function name	What it will do



What buttons will you use, and what will happen when pressed?	Button	What will happen:

Remix Step 4: Write your code

Use the sandbox  when you write the code. Write just a few lines at a time and test often.

Remix Step 5: Commenting and feedback

Documentation	<ul style="list-style-type: none"> • Make sure your code is readable by adding blank lines • Add comments to explain sections of code
---------------	---

Peer feedback: Get feedback from two (or more) people. You can be one of the peer reviewers.

Peer Review #1 Name:

Go through the checklist. Are all requirements met? If not, list any missing criteria.

What do you like about the program – be specific!

Give at least one suggestion. Begin with “what if” or “maybe you could”

Peer Review #2 Name:

Go through the checklist. Are all requirements met? If not, list any missing criteria

What do you like about the program – be specific!

Give at least one suggestion. Begin with “what if” or “maybe you could”

Review the comments. Then take time to improve or add to your project.



Post-Mission Reflection

What did you change in your project after reading the feedback?

What is something new you learned from completing this project?

Unit 3 Remix Rubric Checklist:

- Filename is descriptive
- Uses one or more variables, each with a descriptive name
- Moves the CodeBot forward and/or backward one or more times
- Turns the CodeBot one or more times
- Turns on one or more LED lights
- Uses one or two buttons as input
- Uses at least one sensor to control the CodeBot
- Defines at least one function with a return
- Includes something extra (sound, more than one sensor, more than one function, etc.)
- Code follows programming conventions of comments, readability, indenting, and capitalization
- Code runs with no errors



Unit 4: Underneath the Hood (8-14 hours)

Students learn about internal and sophisticated sensors on CodeBot. First, they get behind the wheels by learning about and programming the wheel encoders for precise driving control. Then they learn about internal sensors for checking battery voltage and CPU temperature. Finally, students learn about the accelerometer, what it is, and an application for detecting movement.

Summary of Mission 8:

In this mission students will learn to navigate their ‘bot using the wheel encoders for specific direction, distance and speed. They will learn what the encoders are and how to use them in code to determine distance, and then for speed and turning angles. This is a fairly detailed lesson with more advanced concepts, including lists, functions with parameters and math calculations.

Summary of Mission 9:

The CodeBot has internal sensors that can measure battery voltage and CPU temperature. Students learn how to access the built-in functions and use them in a meaningful way. The ‘bot’s accelerometer is also introduced. Students practice writing code to understand the data and complete the mission by creating an alarm robot.

Preparation and Materials:

- Students need a computer / laptop with the Chrome web browser.
- Make sure the students can successfully login to <http://make.firialabs.com>
- Students login with their account (possibly using their gmail account)
- Each student (or pair) needs a CodeBot and connecting cable.
- Different floor surfaces & something for an incline, like a sign board or large book
- Optional: something to heat the ‘bot (like a hairdryer) and something to cool the ‘bot (like an ice pack)

Assessment:

Mission 8 Obj. 1-6 Review	Mission 8 Obj. 7-12 Review	Mission 9 Obj 1-7 Review	Mission 9 Obj 8-12 Review
Unit 4 Vocab Review	Unit 4 Coding Review	U4 Vocab Test (MS form)	U4 Coding Test (MS Form)

Standards addressed in this unit:

CSTA Standards Grades 6-8	CSTA Standards Grades 9-10	CSTA Standards Grades 11-12
<ul style="list-style-type: none"> • 2-CS-01 • 2-CS-02 • 2-CS-03 • 2-DA-08 • D-DA-09 • 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-13 • 2-AP-14 • 2-AP-15 • 2-AP-16 • 2-AP-17 • 2-AP-18 • 2-AP-19 	<ul style="list-style-type: none"> • 3A-CS-01 • 3A-CS-02 • 3A-CS-03 • 3A-DA-11 • 3A-DA-12 • 3A-AP-13 • 3A-AP-14 • 3A-AP-15 • 3A-AP-16 • 3A-AP-17 • 3A-AP-18 • 3A-AP-19 • 3A-AP-21 • 3A-AP-23 • 3A-AP-22 • 3A-AP-26 	<ul style="list-style-type: none"> • 3B-CS-02 • 3B-DA-05 • 3B-DA-06 • 3B-DA-07 • 3B-AP-10 • 3B-AP-12 • 3B-AP-14 • 3B-AP-15 • 3B-AP-16 • 3B-AP-17 • 3B-AP-21 • 3B-AP-22 • 3B-AP-23



Mission 8: Navigation	Time Frame: 4-6 hours								
<p>Project Goal: Students will write code to make the CodeBot move exact distances and angles.</p> <p>Learning Targets</p> <ul style="list-style-type: none"> • I can get to know the wheel encoders. • I can calculate speed in cm/sec. • I can create a function to move CodeBot an exact distance using the wheel encoders. • I can write a function to rotate CodeBot using the wheel encoders. 	<p>Key Concepts</p> <ul style="list-style-type: none"> • Each wheel encoder consists of a disc with slots that allow IR light to go through. Counting the light-dark and dark-light intervals can be used to track distance exactly. • Knowing the distance can enable the ‘bot to move precisely a given distance at a given speed. • The encoders can also be used to calculate a turning angle. • Using lists for the data helps manage the information for both wheels. • The time library has built-in functions that can be used to track time 								
<p>Assessment Opportunities</p> <ul style="list-style-type: none"> • Quiz after Objective 3 • Have students explain how the wheel encoders work • Review lists and give sample code for students to work through • Obj. 1-6 Review Kahoot • Obj. 7-14 Review Kahoot • Submit final “EncoderTest” program <p>Supplemental Lab Data Sheets (see appendix)</p> <ul style="list-style-type: none"> • Obj 9 Speedometer (part 1 and part 2) • Surface Test • Data Sheet • Speedometer (procedure 1 and 2) 	<p>Success Criteria</p> <ul style="list-style-type: none"> <input type="checkbox"/> Write code to measure each wheel’s distance traveled. <input type="checkbox"/> Define a drive() function to move CodeBot an exact distance. <input type="checkbox"/> Track distance over time, to measure the speed of the ‘bot’s wheels. <input type="checkbox"/> Calculate the top speed in centimeters per second. <input type="checkbox"/> Write “cruise control” code to maintain a set speed over any terrain. <input type="checkbox"/> Define a rotate() function to turn the ‘bot, and that builds on the encoder code. 								
<p>Vocabulary</p> <ul style="list-style-type: none"> • Wheel encoders: A disc with slots that rotates with a wheel so that an IR light beam can pass through its slots. The pulses of light can be counted to see how much the wheel has rotated. • State: Property of an object; for example True or False. The state can be stored in a variable so the current state can be compared to the previous state. • Speed: Distance / Time (can by any measures; we will use cm/second and also counts/second) • Iterative process: Repeatedly taking small steps to build a whole solution. • Closed loop control: Automates control of a system by sensing the output state and comparing it to the desired state (input). • Feedback loop: Continuously adjusts the system to keep the error, or difference between input and output, close to zero. In our mission, the feedback comes from the encoders, the input is the desired speed and the output is the actual speed. Disturbance can be friction, surface type, etc. • Breakpoint: A marker you can place on any executable line of code that will cause the debugger to stop. 									
<p>New Python Code</p> <table border="1"> <tr> <td><code>val = enc.read(side)</code></td> <td>Reads the encoder’s analog value (LEFT or RIGHT)</td> </tr> <tr> <td><code>if enc_state != slot:</code></td> <td>The != is the “not equals” comparison operator</td> </tr> <tr> <td><code>enc_count = [0, 0]</code></td> <td>Define a list of counters, initializing them to 0</td> </tr> <tr> <td><code>enc_count[LEFT] = enc_count[LEFT] + 1</code></td> <td>Increment a list counter</td> </tr> </table>		<code>val = enc.read(side)</code>	Reads the encoder’s analog value (LEFT or RIGHT)	<code>if enc_state != slot:</code>	The != is the “not equals” comparison operator	<code>enc_count = [0, 0]</code>	Define a list of counters, initializing them to 0	<code>enc_count[LEFT] = enc_count[LEFT] + 1</code>	Increment a list counter
<code>val = enc.read(side)</code>	Reads the encoder’s analog value (LEFT or RIGHT)								
<code>if enc_state != slot:</code>	The != is the “not equals” comparison operator								
<code>enc_count = [0, 0]</code>	Define a list of counters, initializing them to 0								
<code>enc_count[LEFT] = enc_count[LEFT] + 1</code>	Increment a list counter								



<pre>import math CIRCUM = math.pi * DIAMETER</pre>	Import the math library Use math.pi in a calculation
<pre>start_count = enc_count</pre>	Makes a new REFERENCE to the list (not used!)
<pre>start_count = enc_count.copy()</pre>	Makes a copy of the list, each one is separate
<pre>buttons.was_pressed(0)</pre>	Put in code to debounce the button
<pre>import time t_start = time.time()</pre>	Import the time library; ticks_ms marks the passage of time in milliseconds, starting at 0 when the 'bot boots. It keeps counting while the device is running.
<pre>ticks_diff(end_time, start_time)</pre>	Gives the difference between the start and stop time. Use instead of subtraction because like a clock, time can wrap around.
<pre>err = (input - output) * FEEDBACK-PWR Power[side] = power[side] + err</pre>	Calculate the feedback Apply feedback to system
<pre>sleep_ms()</pre>	Delay in milliseconds
<pre>def drive(cm, speed, dir=[+1, +1])</pre>	Set a default parameter

Real World Applications

Rotary encoders are used in any device with a “knob,” like vehicles and appliances. A car odometer and speedometer must use these same calculations to display the car’s distance traveled and speed. A 3D printer’s control system uses a control loop to measure distance and speed to ensure a good print.

Teacher Notes

- Several Lab Data Sheets are available for the mission. Use the ones that are helpful for your students.
- Once again, there is a lot of information given in this lesson, and sometimes the steps go quickly. Take your time with each objective and review the concepts frequently.
- There is a lot of math in this lesson. Review important concepts as needed, and maybe even do some practice problems on paper to help with the understanding and application. The basic distance = rate * time and its variations is the main calculation, but students also need to be familiar with circles and their calculations.

Extensions / Cross-Curricular

- Revisit the “NavSquare” project from Mission 3 and rework the code using wheel encoder code.
- Add sounds and LEDs to the code, indicating what the 'bot is doing.
- Use the buttons to switch between different projects. For example, if BTN-O is pressed, follow a line, but if BTN-1 is pressed, use the wheel encoder code.
- **LANGUAGE ARTS:** Have students write a short description of the 'bot and its wheel encoder capabilities for an online shopping store.
- **SCIENCE:** Encoders use IR light. Have a lesson about light, how it is emitted, reflected and detected.
- **MATH:** The lesson uses the geometry of circles. Have a lesson or activities that involve circles, diameters, and circumference.
- **MATH:** The lesson calculates a turn ratio. Discuss ratios. Or even calculate the distance traveled by the 'bot while it is turning. Or calculate the angle turned given a ratio.



Mission 9: All Systems Go!		Time Frame: 2-3 hours											
<p>Project Goal: Students will use input sensors to monitor battery voltage, system temperature and physical orientation.</p> <p>Learning Targets</p> <ul style="list-style-type: none"> • I can use internal sensors to monitor battery voltage. • I can use internal sensors to monitor system temperature. • I can use internal sensors to monitor physical orientation. • I can use physical orientation to detect motion. 		<p>Key Concepts</p> <ul style="list-style-type: none"> • The ‘bot can measure its own battery voltage and CPU temperature. The data can be used to show alerts to avoid problems. • The accelerometer detects orientation in three dimensions. The CodeBot can be programmed to act on conditions based on its orientation. • The accelerometer can also be used to detect motion in the CodeBot. Any slight movement can be detected and used for an alarm. 											
<p>Assessment Opportunities</p> <ul style="list-style-type: none"> • Quiz after Objective 10 • Submit “BatteryTest” program • Submit “TemperatureCheck” program • Have students give details on the data from the accelerometer. • Submit “AccelTest” program • Obj. 1-7 Review Kahoot • Obj. 8-12 Review Kahoot • Submit final “GuardBot” program <p>Supplemental Lab Data Sheets (see appendix)</p> <ul style="list-style-type: none"> • Obj 3 & 6 (Battery & Temperature Check) • Obj 3 Battery Check • Obj 6 Temperature Check 		<p>Success Criteria</p> <ul style="list-style-type: none"> <input type="checkbox"/> Code a battery tester to tell how much voltage is left in the ‘bot’s battery pack. <input type="checkbox"/> Use the temperature sensor to show an alert to turn on the fan or heater. <input type="checkbox"/> Detect orientation with the accelerometer and rotate the ‘bot toward the sky. <input type="checkbox"/> Make a motion alarm guard-bot. 											
<p>Vocabulary</p> <ul style="list-style-type: none"> • Under load : When batteries are being used to power a peripheral, like turning on LEDs or running motors • User interface: The UI; the part of the computer that humans interact with directly. On the ‘bot it can be buttons and LEDs. • Ambient: Surroundings • Baseline data: Starting point used for comparison; original data • Deadband: In a control system, the range or band of input values where the output doesn’t change; similar to a threshold • Accelerometer: A tiny chip that measures the force of acceleration in three directions: x, y and z • MEMS: Micro-Electro-Mechanical System; a chip with tiny silicon structures inside that really move, with electronic components to sense them 													
<p>New Python Code</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="background-color: #FFF9C4; padding: 5px;"><code>v = system.pwr_volts()</code></td> <td style="padding: 5px;">Measure power supply voltage (either battery or USB) – returns the float power supply voltage</td> </tr> <tr> <td style="background-color: #FFF9C4; padding: 5px;"><code>system.pwr_is_usb()</code></td> <td style="padding: 5px;">Returns 1 for USB (or True) or 0 for battery pack (or False)</td> </tr> <tr> <td style="background-color: #FFF9C4; padding: 5px;"><code>leds.pwr(True)</code></td> <td style="padding: 5px;">Turns on the red LED just above the power switch</td> </tr> <tr> <td style="background-color: #FFF9C4; padding: 5px;"><code>samples.append(temp)</code></td> <td style="padding: 5px;">Adds a value to the end of a list</td> </tr> <tr> <td style="background-color: #FFF9C4; padding: 5px;"><code>samples.clear()</code></td> <td style="padding: 5px;">Empties the list</td> </tr> </table>				<code>v = system.pwr_volts()</code>	Measure power supply voltage (either battery or USB) – returns the float power supply voltage	<code>system.pwr_is_usb()</code>	Returns 1 for USB (or True) or 0 for battery pack (or False)	<code>leds.pwr(True)</code>	Turns on the red LED just above the power switch	<code>samples.append(temp)</code>	Adds a value to the end of a list	<code>samples.clear()</code>	Empties the list
<code>v = system.pwr_volts()</code>	Measure power supply voltage (either battery or USB) – returns the float power supply voltage												
<code>system.pwr_is_usb()</code>	Returns 1 for USB (or True) or 0 for battery pack (or False)												
<code>leds.pwr(True)</code>	Turns on the red LED just above the power switch												
<code>samples.append(temp)</code>	Adds a value to the end of a list												
<code>samples.clear()</code>	Empties the list												



<pre>now = accel.read() x, y, z = accel.read()</pre>	Read the three values from the accelerometer (will be a tuple)
<pre>accel.dump_axes()</pre>	Prints the accelerometer reading on the REPL console
<pre>dx = now[0] - before[0]</pre>	Calculate the difference between the current reading and the previous reading
<pre>if abs(dx) > SENS: alarm()</pre>	If the difference between readings is more than the sensitivity, sound the alarm

Real World Applications

You already use this kind of code daily! Your phone tracks and displays its battery usage. Electronic thermostats control temperature in most buildings. Accelerometers are used in everything from smart watches to game controllers.

Teacher Notes

- In Obj. 3, students will delete a function they created in Obj. 2.
- This mission features several short programs. You may want to do one a day, and spread them out over several days. You can take time to discuss the concepts of each lesson.
- Lab Data Sheets are provided for the battery check and the temperature check. Use the ones that are most helpful for your students.
- You may want to have some extra supplies on hand for some of the missions:
 - Batteries in different stages of use
 - Something to heat up and cool down the bot (like a hair dryer, ice pack, etc.)
 - An incline for the robot to drive on (like a sign board or large book)

Extension / Cross-Curricular

- Combine the battery tester with the temperature check and have the 'bot check use all of its internal sensors in one program.
- Add LEDs to the guard bot program
- Add sound to the battery check and temperature check programs.
- Use a button to turn on/off the guard bot.
- **LANGUAGE ARTS:** Have students write a lab report or technical paper about one of the programs in the mission.
- **MATH:** The lesson uses the $y=mx+b$ equation for battery percentage. Plot your own points from battery data and create your own equation. Or use the equation for other data.
- **SCIENCE:** The accelerometer measures gravity. Have a lesson on gravity, and maybe in space and on other planets.
- **SCIENCE:** The first program talks about the "load" of a battery. Create some simple circuits and measure the loads for each one.



Unit 4 Remix Project and Exam	Time Frame: 2-5 hours
<p>Remix Project Goal: Students will use the skills and concepts they learned in Mission 8 and Mission 9 to create their own project.</p> <p>Remix Project Outline: Follow the five-steps of the design process (document on next page) to design a remix project.</p>	<p>Remix Assessment Opportunities</p> <ul style="list-style-type: none"> ● Peer reviews / Gallery walk ● Remix 4 Log planning guide ● Remix Rubric Checklist ● Submit Remix Program <p>Unit 4 Exam Opportunities</p> <ul style="list-style-type: none"> ● Unit 4 Vocabulary review Kahoot ● Unit 4 Concepts and coding review Kahoot ● Unit 4 Vocabulary test (MS Form) ● Unit 4 Concepts & coding test (MS Form)
<p>Remix Project Ideas:</p> <ul style="list-style-type: none"> ● Pick an extension idea from Mission 8 or Mission 9. ● Combine the animatronics with the motion sensor. Have the robot before some animation when motion is detected (like a spooky Halloween character). ● Combine the battery tester with the navigation program so that you are warned when the battery power is low during a driving mission. ● Combine the accelerometer readings with the navigation system for specific driving capabilities. ● Think of your own creative project with line or proximity sensors, movement, LEDs and button input. 	
<p>Remix Rubric Checklist:</p> <ul style="list-style-type: none"> <input type="checkbox"/> Uses one or more variables, each with a descriptive name <input type="checkbox"/> Reads one or more sensors: encoders, battery power, temperature, or accelerometer <input type="checkbox"/> Uses the data from the sensor reading to control the CodeBot <input type="checkbox"/> Controls one or more peripherals: LEDs, sound, motors <input type="checkbox"/> Uses one or two buttons as input <input type="checkbox"/> Defines and uses at least one function <input type="checkbox"/> Defines and uses at least one list <input type="checkbox"/> Includes something extra (sound, more than one sensor, more than one function, etc.) <input type="checkbox"/> Code follows programming conventions of comments, readability, indenting, and capitalization <input type="checkbox"/> Code runs with no errors 	




Unit 4 Remix Log	Name:													
Remix Step 1: Review your code from Mission 8 and 9														
Mission 8: Navigation What does this program do?														
Mission 9: All Systems Go! What does this program do?														
What programming concepts did you learn and use in each mission?														
Remix Step 2: Remix Project Concept														
Look over the remix suggestions. Discuss with a partner. Then decide what you want to do for your remix project. Describe what your remix project will do:														
Remix Step 3: Plan your code. What variables will you use in the project? Fill out the charts below. Use another piece of paper to design your program with a flowchart or pseudocode.														
What variables and lists will you use in the project? Fill in the chart. You do not need to fill in every line, or you can add more.	<table border="1"> <thead> <tr> <th>Variable / List Name</th> <th>What it will be used for:</th> </tr> </thead> <tbody> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </tbody> </table>		Variable / List Name	What it will be used for:										
	Variable / List Name	What it will be used for:												
What functions will you write? Describe each one.	<table border="1"> <thead> <tr> <th>Function name</th> <th>What it will do</th> </tr> </thead> <tbody> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </tbody> </table>		Function name	What it will do										
	Function name	What it will do												



What buttons will you use, and what will happen when pressed?	Button	What will happen:

Remix Step 4: Write your code

Use the sandbox  when you write the code. Write just a few lines at a time and test often.

Remix Step 5: Commenting and feedback

Documentation	<ul style="list-style-type: none"> • Make sure your code is readable by adding blank lines • Add comments to explain sections of code
---------------	---

Peer feedback: Get feedback from two (or more) people. You can be one of the peer reviewers.

Peer Review #1 Name:

Go through the checklist. Are all requirements met? If not, list any missing criteria.

What do you like about the program – be specific!

Give at least one suggestion. Begin with “what if” or “maybe you could”

Peer Review #2 Name:

Go through the checklist. Are all requirements met? If not, list any missing criteria

What do you like about the program – be specific!

Give at least one suggestion. Begin with “what if” or “maybe you could”

Review the comments. Then take time to improve or add to your project.



Post-Mission Reflection

What did you change in your project after reading the feedback?

What are some reasons you would give your friend for learning to program?

Unit 4 Remix Rubric Checklist:

- Uses one or more variables, each with a descriptive name
- Reads one or more sensors: encoders, battery power, temperature, or accelerometer
- Uses the data from the sensor reading to control the CodeBot
- Controls one or more peripherals: LEDs, sound, motors
- Uses one or two buttons as input
- Defines and uses at least one function
- Defines and uses at least one list
- Includes something extra (sound, more than one sensor, more than one function, etc.)
- Code follows programming conventions of comments, readability, indenting, and capitalization
- Code runs with no errors



Mission Pack Final Project	Time Frame: 4-8 hours
<p>Final Project Goal: Students will use the skills and concepts they learned during the mission pack “Python with Robots” to create their own project.</p> <p>Final Project Outline: Follow the five-steps of the design process (document on next page) to design a final project.</p>	<p>Assessment Opportunities</p> <ul style="list-style-type: none"> ● Final project planning document (with rubric) ● CodeBot Project Rubric (CSTA Standards) ● Peer reviews / Gallery walk ● Remix Rubric Checklist ● Submit Final Program ● Student Reflection ● Project presentation or report
<p>Final Project Ideas:</p> <ul style="list-style-type: none"> ● Program the CodeBot to run an obstacle course, avoiding obstacles and navigating the course. ● Program the CodeBot to multitask – perform more than one task at a time. For example, it can navigate a square while blinking an LED ten times per second. ● Think of your own creative project that combines concepts and code from several of the missions into something unique and that you find interesting. 	
<p>Teacher Notes:</p> <ul style="list-style-type: none"> ● If you have your students try programming the ‘bot for the obstacle course, it can be an interesting and fun class competition. ● A rubric is provided, but feel free to adjust it to the needs and special interests of your students. ● Most state and national computer science standards include teamwork and time management. Use the final project as an opportunity for teamwork, leadership roles, electronic communication and managing a project. Review the computer science standards for your state and grade band, and incorporate them into this project. CSTA Standards are listed below. <ul style="list-style-type: none"> ○ Grades 6-8 ○ 2-AP-18: Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts. ○ 2-IC-22: Collaborate with many contributors through strategies such as crowdsourcing or surveys when creating a computational artifact. ○ Grades 9-10 ○ 3A-AP-22: Design and develop computational artifacts working in team roles using collaborative tools. ○ 3A-AP-27: Use tools and methods for collaboration on a project to increase connectivity of people in different cultures and career fields. ○ Grades 11-12 ○ 3B-AP-20: Use version control systems, integrated development environments (IDEs) and collaborative tools and practices (code documentation) in a group software project. ● Most state and national computer science standards also include evaluating computational artifacts. Use the final project as a point of discussion for the global impact of computers. The standards are listed below to help guide class discussions, written prompts, project requirements, etc. <ul style="list-style-type: none"> ○ 2-IC-20: Compare tradeoffs associated with computing technologies that affect people’s everyday activities and career options. ○ 2-IC-21: Discuss issues of bias and accessibility in the design of existing technologies. ○ 3A-IC-24: Evaluate the ways computing impacts personal, ethical, social, economic and cultural practices. ○ 3A-IC-25: Test and refine computational artifacts to reduce bias and equity deficits. ○ 3B-IC-25: Evaluate computational artifacts to maximize their beneficial effects and minimize harmful effects on society. ○ 3B-IC-26: Evaluate the impact of equity, access, and influence on the distribution of computing resources in a global society. ○ 3B-IC-27: Predict how computational innovations that have revolutionized aspects of our culture might evolve. 	




Final Project Planning	Name:											
Other team members:												
Remix Step 1: Review your code from the mission pack												
What programs / missions were your favorite? What did you like about them?												
What programming concepts do you feel you understand the most?												
What programming concepts do you need help with?												
Remix Step 2: Final Project Concept												
Look over the remix suggestions from your favorite projects. Discuss with your team. Then decide what you want to do for the final project that will both interest and challenge you. Describe what your final project will do:												
Remix Step 3: Plan your code. What variables will you use in the project? Fill out the charts below. Use another piece of paper to design your program with a flowchart or pseudocode.												
What variables and lists will you use in the project? Fill in the chart. You do not need to fill in every line, or you can add more.	<table border="1"> <thead> <tr> <th>Variable / List Name</th> <th>What it will be used for:</th> </tr> </thead> <tbody> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </tbody> </table>		Variable / List Name	What it will be used for:								
	Variable / List Name	What it will be used for:										



What functions will you write? Describe each one.	Function name	What it will do

What buttons will you use, and what will happen when pressed?	Button	What will happen:

Remix Step 4: Write your code

Use the sandbox  when you write the code. Write just a few lines at a time and test often.

Remix Step 5: Commenting and feedback

Documentation	<ul style="list-style-type: none"> • Make sure your code is readable by adding blank lines • Add comments to explain sections of code
---------------	---

Peer feedback: Get feedback from two (or more) people.

Peer Review #1 Name:	
Go through the rubric. Are all requirements met? If not, list any missing criteria.	
What do you like about the program – be specific!	
Give at least one suggestion. Begin with “what if” or “maybe you could”	



Peer Review #2 Name:	
Go through the rubric. Are all requirements met? If not, list any missing criteria	
What do you like about the program – be specific!	
Give at least one suggestion. Begin with “what if” or “maybe you could”	

Review the comments. Then take time to improve or add to your project.

Post-Mission Reflection

What did you change in your project after reading the feedback?	
What do you like most about programming?	
What do you find the most challenging about programming?	
How have your attitudes or feelings about computer science changed during this mission pack?	



Final Project Rubric			
Requirement	No evidence ←-----→ Mastery		
Programming Conventions are followed	<ul style="list-style-type: none"> • Variable names aren't descriptive • Function names aren't descriptive • Code blocks inconsistently indented • Capital letters used • Code is not organized into sections 		<ul style="list-style-type: none"> • Variable names are descriptive • Function names are descriptive • Code blocks consistently indented • Use of small letters (not capital) • Code is organized into sections
Documentation and Readability	<ul style="list-style-type: none"> • No comments are used. • Code is difficult to read because no blank lines were used, or too many blank lines were included. 		<ul style="list-style-type: none"> • Frequent and descriptive comments are used regularly. • Blank lines are used to help with readability.
Use of Variables and constants	<ul style="list-style-type: none"> • "Magic Numbers" or literal values are used in the code. • Data isn't tracked or updated (no counters, states, conversions, etc.). 		<ul style="list-style-type: none"> • Constants are used to eliminate "magic numbers." • Variables are used for storing, keeping track of and updating data. • Global and local variables are used.
Use of Functions	<ul style="list-style-type: none"> • No plan or algorithm to follow. • Everything in one main program. • Long sections of code. • Functions use all global or all local variables. • Functions don't take parameters. 		<ul style="list-style-type: none"> • Code is divided into smaller sections that accomplish a task. • Parameters are used as needed. • Local and global variables are used as needed. • Functions return a value as needed.
Use of Inputs Buttons and sensors	<ul style="list-style-type: none"> • Neither button is used for input. • No sensors are read or used. (line sensor, proximity sensor, encoders, system temperature, battery voltage, accelerometer) 		<ul style="list-style-type: none"> • At least one button is used for input and control. • At least one sensor is used to give input. • Conversion of raw data is performed as needed.
Algorithms and Programming	<ul style="list-style-type: none"> • No algorithms identified or used. • Program performs the same for every execution, without input. • Lists and tuples are not utilized when they would simplify the code. • Debugging practices are not used and code contains errors. 		<ul style="list-style-type: none"> • Algorithms are used to manipulate data and get results. • Data is used to inform decisions. • Lists and tuples are used to simplify data collection and implementation. • Debugging practices are used to correct errors in code and logic.
Control Structures	<ul style="list-style-type: none"> • Program does not have any if or if/else or if/elif/else statements. • Program does not use any while loops. • Nested loops or if statements are not used, or are used incorrectly. 		<ul style="list-style-type: none"> • While loops and if statements are used to control the flow of execution. • Conditional and logical operators are used appropriately. • Nested while and if statements are used when needed.
Use of Outputs LEDs, speaker, motors	<ul style="list-style-type: none"> • No output is produced. 		<ul style="list-style-type: none"> • One or more outputs are used to convey data or perform a task.
Collaboration	<ul style="list-style-type: none"> • Students work independently or uncooperatively on a team. 		<ul style="list-style-type: none"> • Students work collaboratively with shared tasks in their team to complete the project.
Synthesis / Purpose	<ul style="list-style-type: none"> • No clear purpose for the program. • Program does not incorporate learning across the mission pack. 		<ul style="list-style-type: none"> • Purpose of the program is clearly stated. • Program combines learning, concepts and code from several missions.
Code Completion	<ul style="list-style-type: none"> • Code will not run or doesn't complete the task correctly. 		<ul style="list-style-type: none"> • Code runs and accomplishes its task without any errors, including logic.



Appendix A: Required Resources

Computer Resources

Each student will need:

- A computer with the Chrome web browser.
- Chromebooks work great – just make sure they are up to date.
- Windows 10 or Windows 11 will work with no additional drivers needed.
- A current Mac OS will also work with no additional drivers needed.
- A USB port is used to connect and program the CodeBot. The CodeBot comes with a USB to USB-C cable. If your laptop or computer has any other configuration, you will need a cable that has USB-C on one end.

Software Resources

- The interactive textbook and text editor is web-based. Make sure the website is not blocked.
- An email is required for signing in and saving work. It can be a gmail account, but any email will work.
- A per CodeBot device license is needed to access the curriculum.

Physical Resources

The missions can be completed by individual students or student pairs utilizing pair programming. It is possible to share a CodeBot with more than one student or student pair in the same class, but that is not recommended. Each student or student pair will still need a CodeBot and license for the curriculum.

All CodeBots and curriculum licenses can be used throughout the day with different classes and groups of students.

The CodeBot comes with a connecting USB cable and curriculum license. Other materials that will be needed throughout the missions are:

- 4 AA batteries for each CodeBot.
- Black tape (or something similar) for creating lines on a white surface. Alternately, you can use light-colored tape on a dark surface.
- Objects to place in front of the CodeBot for detection.
- Metric measuring stick.
- Different surfaces for friction. This could be tile, carpet, paper, white boards, etc.
- Different surfaces for reflection. They can be black, white and colored paper, or anything similar.
- Something to heat and cool the 'bot (like a hair dryer and ice pack).

Notes

- When the CodeBot is plugged into a computer, it will appear as a USB mass storage device, similar to a flash drive. This is not required for normal classroom use. So don't worry if your school has a policy preventing flash drives. You just close the pop-up window and continue.
- Occasionally Firia Labs will provide a software update that requires updating the core software on the CodeBot. At those times you will need the flash drive feature to update the software, so you will need to use a computer with USB drive access. Often a teacher's computer is used to update all the CodeBot.



Appendix B: Our Approach

Physical Computing and CodeSpace: a web-based professional-learning platform

Hardware brings code to life! Our versatile physical computing devices and peripherals get students excited about code. Our CodeSpace learning environment enables them to step up to computer science with real-world text-based Python coding. We include ready-to-teach standards-aligned curriculum with hands-on projects that motivate students.

While there are some great online coding educational programs, we think our approach helps reach a broader range of students. Our approach:

- Gets students focused “off-screen,” programming with physical hardware that connects and interacts independently of their computers.
- Teaches a real, professional programming language. Even younger students appreciate that you can make real money with these exact skills. If they can read, and they can type, they can code in text-based Python.
- Gives students the tools to create *anything* they can imagine. Beyond projects and curriculum, we give students a full-fledged software development environment. These are professional-strength tools for writing code. Instead of a game-playing environment, students can “win with code” through engaging hands-on projects and their own creativity.

Project Based Motivation

Students may wonder why they are learning to code. We all find that knowledge tastes so much better when you’re hungry for it! Our goal is to **motivate** students with tangible, challenging and practical **projects**...that just so happen to require coding to build. We want students to think about how they might code a given project using what they already know. Only then do we teach *just enough* coding concepts to help them get the job done. This approach gives reason and meaning to each concept, as well as relevant problem context, which helps them retain it.

Type it In

Students are often tempted to just copy and paste from lesson examples. Prior to our extensive testing of the curriculum on groups of 4th through 12th grade students, we were concerned that the typing burden might be a problem. But we were willing to risk it.

- Typing in the code forces focus, dramatically improving retention.
- Keyboarding proficiency is key to expressiveness in using a programming language.
- Mistakes in structure, grammar, punctuation, capitalization, etc. are priceless learning opportunities.

Students learn an incredible amount from their mistakes. Our goal is to provide awesome safety-nets for them, guiding them to iterate quickly through successive failed attempts to arrive at a working solution. Extensive classroom observation has convinced us that the typing burden is not a problem. Students dive right in, and they don’t have to be speed typists to make great progress in coding.

Exploration and Creativity

One of the great things about coding is the expressiveness it affords. Coding is a craft that takes time to master, but with only a few basic tools you can start crafting some pretty amazing things! Before they even complete the first project, some of your students will probably be experimenting “off-script” with some ideas of their own. That’s a good thing! In every lesson we list some ideas for re-mixing each project’s concepts. Remember that students are learning programming skills they can use to build *any* application – from controlling a rocketship to choreographing dance moves. Nurture creativity, explore, and instill the joy of coding!



Appendix C: Teacher Resources

If you and your students are still fairly new to text-based coding, don't worry! Like other physical devices and their curriculum, we've designed the Python with Robots Mission Pack and this curriculum guide to gently guide you from absolute beginner to a very comfortable level of proficiency. Remember this – Don't Panic 😊

We understand that tackling a subject like Computer Coding can be pretty intimidating. Fear not, we've built some amazing tools to help you! As you begin this journey, know that the team at Firia Labs is here to help, too. If you run into any problems, just let us know and we'll get you back on track.

Classroom Preparation

Writing code can be like literary writing. Like developing writing skills requires individual practice, learning to code requires students to compose and test their work individually. They need to make their own mistakes and struggle through correcting them.

There is also a place for pair programming and collaboration in the coding classroom. Such practices foster knowledge sharing, collective code ownership and code review “on the go”. It also gives students a chance to communicate about what they are learning and reflect on their practices. It builds confidence and keeps students focused on the task. Pair programming can result in better quality work with less errors, and keeps teams “in the flow”.

You may need to think about a balance between independent work and pair programming to give your students the best opportunities to succeed and truly engage in and enjoy programming.

Daily Routine

We recommend students work for at least 30 minutes each programming session. Adjust accordingly to your day. Because of the time it takes to set up equipment, log in to computers, and then collect equipment at the end of the learning period, it may take more time than you anticipate. Each lesson has a suggested time frame. This range accounts for completing the basics to continuing with cross-curricular lessons or extensions. Some missions may go even longer, depending on the time you have to spend in coding, the length of time for each mission, the abilities of your students, etc.

This mission pack has a lot of flexibility built-in. You should complete each mission in order, but the amount of time spent on each mission is up to you. A pacing calendar isn't provided, given the flexibility and options for the mission pack. But a suggested timeframe for each lesson is given to help you decide how best to plan for the course in your class period with your seat minutes, ability level, other constraints, etc.

We recommend that students complete the Python with CodeX mission pack in advance, but it is not required. For pacing considerations, the mission pack can be:

- A once-a-week activity for an elective class or after school club
- A drop-in unit in a required or elective course
- Extended to a 9-week or 18-week course

Extensions

Naturally students will progress at different speeds. The material is set up for independent study. You can allow students to work ahead at their own pace, or slow down as needed.

As an alternative, you can keep the class together and have “high flyers” work on extensions to the missions. Several suggestions are given for each mission.



Extensions give students a chance to review their learning and add to their program in ways that interest them. Many students will want to experiment with what they've learned, and we offer suggestions along the way to spur this creative tinkering. Extensions are also an excellent opportunity for students to synthesize their learning and create their own projects. We highly recommend including extensions into your pacing calendar.

Cross Curricular

Another natural extension to each mission is tie-in the project to other subject areas. Suggestions are given for each mission to extend student learning through a topic in another subject, such as language arts, math and science. These extensions can be separate lessons that build from the mission, lessons given before the mission, or just other ways to look at the project. If you teach multiple subjects or work with teachers in other disciplines, you may want to consider adding in cross curricular extensions as well.

Managing a Class

Our CodeSpace learning platform makes it easy for you to create a class for your students to join, and enables you to monitor their progress.

For help and step-by-step instructions, visit: <https://learn.firialabs.com/curricula/code-space>

If you are a **Google Classroom** teacher, you can import assignments from CodeSpace into your classes. For instructions, go to “Virtual Tools with CodeSpace” in the [Teacher Resources for Python with Robots](#).

If you need assistance for anything, please send an email to: support@firialabs.com

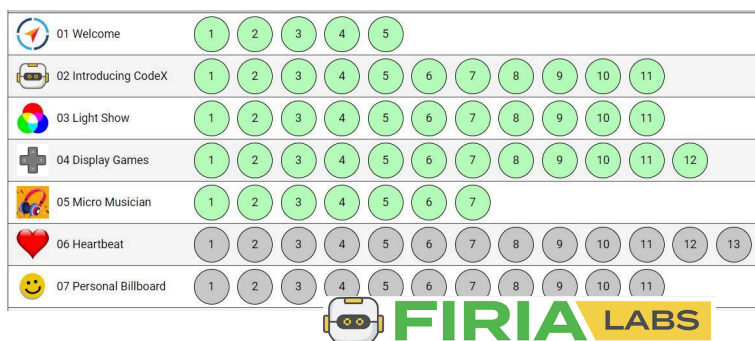
Here are the basics of the CodeSpace Teacher Dashboard

- Log in to CodeSpace and from HELP, select CLASS DASHBOARD
- Once you are in the dashboard, click + in the green bar, top right corner, to add a class.
- Assign each class a name, and allow members to join with a join code.
- You can assign Google Classroom as your LMS.
- After the class is created, you can edit the class, get a join code, disable joining, etc.
- You can delete a student using the “remove” function.
- Students go to CodeSpace and click the SELECT CLASS button.
- They can click the JOIN CLASS button and enter their join code for your class.
- The class will be activated and they are ready to start working!
- In the dashboard, you can see student progress, as a whole class and individually.

Class dashboard

← BACK						My Classes	Coding Cousins
Email ↑	Name	Overa...	XP	Completed Up To	Last Active		
james.egg@gmail.com	Jill Jones	84%	795	Mission 15, Objective 1	-		
james24@pro.net		14%	125	Mission 3, Objective 8	-		
melvin.tyler@gmail.com		29%	260	Mission 6, Objective 1	-		

Individual progress





Appendix D: Assessing Student Projects

The lessons give many opportunities for formative assessment. Any formative assessments you already use in your classroom can be used with programming assignments. Each lesson has suggestions for assessment, including the quizzes embedded in the interactive textbook, turning in completed programs, and Kahoot! Reviews. Each unit has a vocabulary test and a concepts and coding test. Review Kahoot!s are available for each. Also, the remix project for each unit can be used for assessment. A rubric checklist is included for each remix project.

Remix Projects

A generic project rubric is included on the next page. It can be used as a written form, or made into a digital form. A CSTA Standards rubric is also provided on the learning portal (CodeX Project Rubric). Either rubric may need to be modified for earlier projects, since not all standards are met with every mission. Make a copy and edit as needed. You can also customize the rubric by adding custom requirements or assigning point values before students begin.

Students should be given a copy of the rubric before beginning the project. Discuss the criteria and what it means to earn mastery. It is beneficial to give students time to revise and improve their projects, as time permits. Students who approach mastery may be motivated to improve, so decide what your classroom policy and expectations will be and explain it to students early on. You may need to revise policies as you get to know your students and observe how CodeSpace works for them. Flexibility is important!

Student-Teacher Conferencing

Student-teacher conferencing is integral to the learning process. This takes more time in class, but this is not wasted time! Students will work harder and be more willing to do revisions, which is truly a workplace life skill we'd like to instill in our students. To manage the process, it helps to have a submission window, rather than one set due date. Once a student submits their work, call him/her up for a conference. Begin with an open-ended question, like "Tell me about your project." Then move on to the rubric. This may give you insight into who did what, if working in pairs, and what challenges they encountered. As you conference about the rubric, ask them what level of mastery they think they achieved, and why. Students are often more critical of their work than they need to be. It's a good time to emphasize challenges and mistakes are learning opportunities rather than just being "wrong." If time allows, students should be allowed to debug and improve before a final submission of their work.

Peer Feedback

Before students submit a remix project, they should complete a peer review. This may take modeling a few times before students do it correctly. Remind students that revising is just as important here as it is in English class. These revisions can lead to great conversations during the conferencing process. They should go through the rubric and test the program just as you would. This will give them the chance to find and correct mistakes before doing a student-teacher conference. A peer review form is included in the document for every remix. A sample peer review form is included in this appendix.

Early Finishers

Students who finish earlier than the submission deadline may enjoy having time to work on other unscripted projects, and just trying things out. This is not wasted time! Learning through trial and error is time well-spent, and we want to encourage curiosity for their motivation.



Project Rubric			
Requirement	No evidence ←-----→ Mastery		
Programming Conventions are followed	<ul style="list-style-type: none"> Variable names aren't descriptive Function names aren't descriptive Code blocks inconsistently indented Capital letters used Code is not organized into sections 		<ul style="list-style-type: none"> Variable names are descriptive Function names are descriptive Code blocks consistently indented Use of small letters (not capital) Code is organized into sections
Documentation and Readability	<ul style="list-style-type: none"> No comments are used. Code is difficult to read because no blank lines were used, or too many blank lines were included. 		<ul style="list-style-type: none"> Frequent and descriptive comments are used regularly. Blank lines are used to help with readability.
Use of Variables and constants	<ul style="list-style-type: none"> "Magic Numbers" or literal values are used in the code. Data isn't tracked or updated (no counters, states, conversions, etc.). 		<ul style="list-style-type: none"> Constants are used to eliminate "magic numbers." Variables are used for storing, keeping track of and updating data. Global and local variables are used.
Use of Functions	<ul style="list-style-type: none"> No plan or algorithm to follow. Everything in one main program. Long sections of code. Functions use all global or all local variables. Functions don't take parameters. 		<ul style="list-style-type: none"> Code is divided into smaller sections that accomplish a task. Parameters are used as needed. Local and global variables are used as needed. Functions return a value as needed.
Use of Inputs Buttons and sensors	<ul style="list-style-type: none"> Neither button is used for input. No sensors are read or used. (line sensor, proximity sensor, encoders, system temperature, battery voltage, accelerometer) 		<ul style="list-style-type: none"> At least one button is used for input and control. At least one sensor is used to give input. Conversion of raw data is performed as needed.
Algorithms and Programming	<ul style="list-style-type: none"> No algorithms identified or used. Program performs the same for every execution, without input. Lists and tuples are not utilized when they would simplify the code. Debugging practices are not used and code contains errors. 		<ul style="list-style-type: none"> Algorithms are used to manipulate data and get results. Data is used to inform decisions. Lists and tuples are used to simplify data collection and implementation. Debugging practices are used to correct errors in code and logic.
Control Structures	<ul style="list-style-type: none"> Program does not have any if or if/else or if/elif/else statements. Program does not use any while loops. Nested loops or if statements are not used, or are used incorrectly. 		<ul style="list-style-type: none"> While loops and if statements are used to control the flow of execution. Conditional and logical operators are used appropriately. Nested while and if statements are used when needed.
Use of Outputs LEDs, speaker, motors	<ul style="list-style-type: none"> No output is produced. 		<ul style="list-style-type: none"> One or more outputs are used to convey data or perform a task.
Collaboration	<ul style="list-style-type: none"> Students work independently or uncooperatively on a team. 		<ul style="list-style-type: none"> Students work collaboratively with shared tasks in their team to complete the project.
Synthesis / Purpose	<ul style="list-style-type: none"> No clear purpose for the program. Program does not incorporate learning across the mission pack. 		<ul style="list-style-type: none"> Purpose of the program is clearly stated. Program combines learning, concepts and code from several missions.
Code Completion	<ul style="list-style-type: none"> Code will not run or doesn't complete the task correctly. 		<ul style="list-style-type: none"> Code runs and accomplishes its task without any errors, including logic.



Peer Review Form	
Peer Reviewer Name:	
Programmer being reviewed:	
Project being reviewed:	
Go through the rubric. Are all requirements met? If not, list any missing criteria	
What do you like about the program – be specific!	
Give at least one suggestion. Begin with “what if ... or “maybe you could ...”	



Appendix E: Links to teacher materials

Code Solutions	Teacher Resources for Python with Robots (Answer Keys)
Vocabulary by Mission	Teacher Resources for Python with Robots (General Resources)
Python Code by Mission	Teacher Resources for Python with Robots (General Resources)
Unit 1 Review Kahoots and Exams (Mission 2 & 3)	
Mission 2 Review	https://create.kahoot.it/share/firia-labs-codebot-mission-2/2925c213-a0c5-4ed4-8e-fa-45dc8d9db0e7
Mission 3 (Obj 1-6)	https://create.kahoot.it/share/firia-labs-codebot-mission-3-obj-1-6/d53e34d5-56ab-4962-a9e4-a6075bb90954
Mission 3 (Obj 7-9)	https://create.kahoot.it/share/firia-labs-codebot-mission-3-obj-7-9/ced9ca2e-c1c1-4779-8494-68e27eaa52db
Mission 3 (Obj 10-11)	https://create.kahoot.it/share/firia-labs-codebot-mission-3-obj-10-11/68a85b45-3616-4657-ad5d-472632455efd
Unit 1 Vocabulary Review	https://create.kahoot.it/share/firia-labs-codebot-unit-1-vocab-review/0971b640-bebc-4ca7-8745-54f7f814521b (compilation of 13 terms from previous reviews, plus two more terms)
Unit 1 Concepts and Coding Review	https://create.kahoot.it/share/firia-labs-codebot-unit-1-coding-review-missions-1-3/080d70c6-365a-4440-9a95-7bba5b59eaa9
Unit 1 Vocab Test	Microsoft Forms (make a duplicate)
Unit 1 Coding Test	Microsoft Forms (make a duplicate)
All Review & Test Questions	Teacher Resources for Python with Robots (General Resources)
Unit 2 Review Kahoots and Exams (Mission 4-5)	
Mission 4 (Obj 1-5)	https://create.kahoot.it/share/firia-labs-codebot-mission-4-obj-1-5/274224e8-9c7f-42bb-a648-c9b334bb7cfe
Mission 4 (Obj 6-12)	https://create.kahoot.it/share/firia-labs-codebot-mission-4-obj-6-12/1b909d22-067e-4135-ac7f-bba9273c70ad
Mission 5	https://create.kahoot.it/share/firia-labs-codebot-mission-5/20d9499d-fe50-45a7-9f2c-623975832277
Unit 2 Vocab Review	https://create.kahoot.it/share/firia-labs-codebot-unit-2-vocabulary-review-missions-4-5/456e14e6-6a4f-43b5-91b3-d6af75397a6c
Unit 2 Code Review	https://create.kahoot.it/share/firia-labs-codebot-unit-2-coding-review-missions-4-5/c68432d2-742d-4879-b4d6-3378aae37ddb
Unit 2 Vocab Test	Microsoft Forms (make a duplicate)
Unit 2 Coding Test	Microsoft Forms (make a duplicate)
All Review & Test Questions	Teacher Resources for Python with Robots (General Resources)



Unit 3: Missions 6-7	
Mission 6	https://create.kahoot.it/share/firia-labs-codebot-mission-6/efb31058-8e4b-4a7d-8239-32489c5462c9
Mission 7	https://create.kahoot.it/share/firia-labs-codebot-mission-7/d66d1ea8-5156-459d-aea-d3038dc638b6
Unit 3 Vocab Review	https://create.kahoot.it/share/firia-labs-codebot-unit-3-vocabulary-review-missions-6-7/73b634fe-374f-4ed3-98d6-16a2d77c3807
Unit 3 Code Review	https://create.kahoot.it/share/firia-labs-codebot-unit-3-coding-review-missions-6-7/b276f5e7-8e82-479b-bf35-f1a93737b251
Unit 3 Vocab Test	Microsoft Forms (make a duplicate)
Unit 3 Coding Test	Microsoft Forms (make a duplicate)
All Review & Test Questions	Teacher Resources for Python with Robots (General Resources)
Unit 4: Missions 8-9	
Mission 8 (1-6)	https://create.kahoot.it/share/firia-labs-codebot-mission-8-obj-1-6/5af4f55a-fce5-4134-b4bf-e2f8309d3fb3
Mission 8 (7-14)	https://create.kahoot.it/share/firia-labs-codebot-mission-8-obj-7-14/00c0584e-f8ea-4651-a2a3-4db0a88e7a7c
Mission 9 (1-7)	https://create.kahoot.it/share/firia-labs-codebot-mission-9-obj-1-7/fccc2ba6-c3ca-4f6b-a131-44af6af14b80
Mission 9 (8-12)	https://create.kahoot.it/share/firia-labs-codebot-mission-9-obj-8-12/aab9df2a-de3b-47e2-9793-6b7ce15a5ae2
Unit 4 Vocab Review	https://create.kahoot.it/share/firia-labs-codebot-unit-4-vocabulary-review-missions-8-9/759f2751-5c37-4049-b07d-f27ad2704c00
Unit 4 Coding Review	https://create.kahoot.it/share/firia-labs-codebot-unit-4-coding-review-missions-8-9/53f79d01-600a-47e0-87ba-8a9c48d328c9
Unit 4 Vocab Test	Microsoft Forms (make a duplicate)
Unit 4 Coding Test	Microsoft Forms (make a duplicate)
All Review & Test Questions	Teacher Resources for Python with Robots (General Resources)



Appendix F: Lab Data Sheets

Lab Data Sheets for each mission are included in this section. You can also access printer-friendly versions of each document at learn.firialabs.com

Mission 3 Obj. 7 Get Moving	54
Test Surfaces (can be used with several missions)	55
Mission 5 Obj. 2 Line Sensors	56
Mission 5 Obj. 2 Debug Console	57
Mission 6 Obj. 2 REPL	58
Mission 6 Obj. 3, 4, 5	59
Mission 6 Obj. 5 Line Follower Reflection	60
Mission 7 Obj. 1 Presence Detector Experiment	61, 62
Mission 7 Obj. 2, 4, 7	63
Mission 7 Obj. 2 Power	64
Mission 7 Obj. 2 Threshold	65
Mission 7 Reflection	66
Mission 8 Obj. 9 Speedometer	67
Mission 8 Surface Test	68
Mission 8 Wheel Encoder Test	69
Mission 9 Obj 3, 6	70
Mission 9 Battery Test	71
Mission 9 Temperature Test	72



Python with Robots Mission 3 Lab Data Sheet – OBJ 7	Name:
--	--------------

Obj 7 Get Moving: How far does the CodeBot move if both wheels have the same power, with a given delay?

Wheel power setting	Sleep delay	Distance traveled

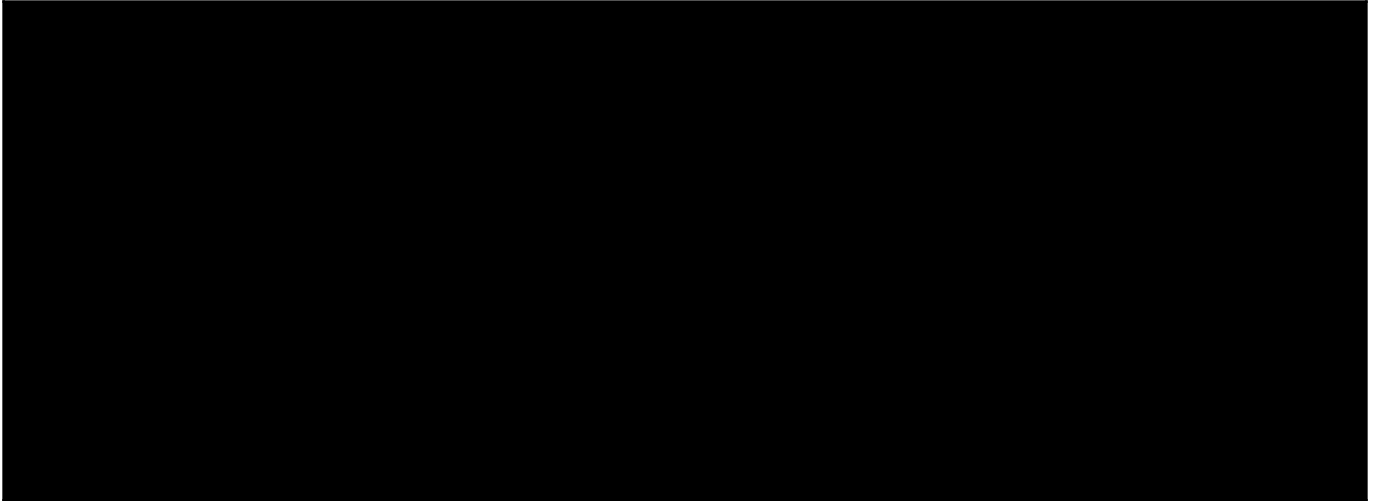
Obj 8 Rotation Time: What angle does the CodeBot turn with a given wheel power and delay?

Left Wheel power	Right Wheel power	Sleep delay	Angle and rotation (which way)

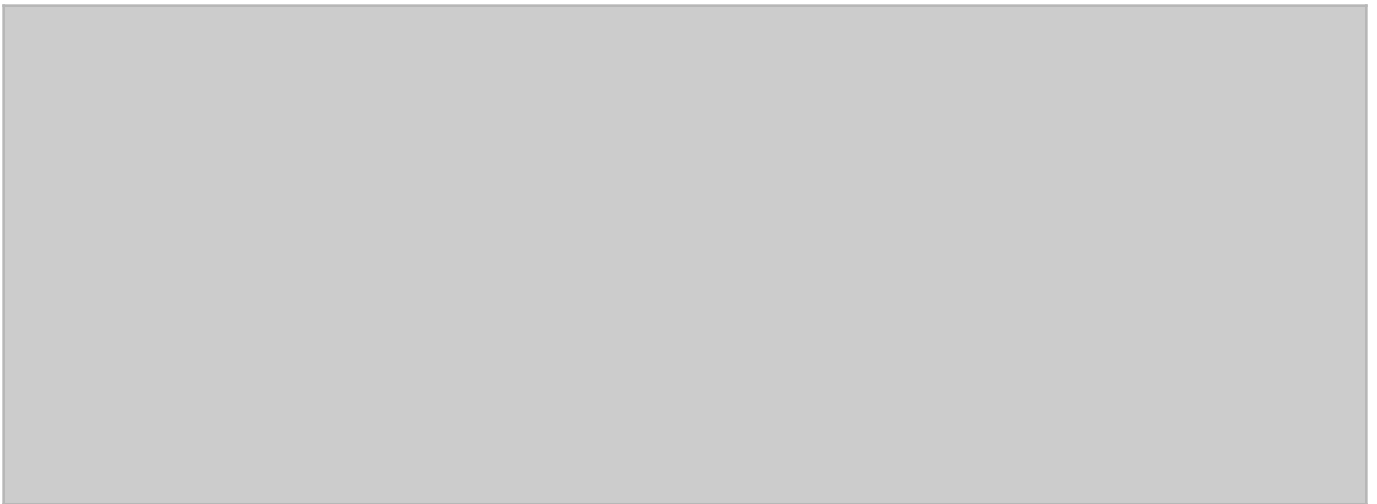
Planning Guide: Turn this paper over. Plan your navigation square by using a flowchart of pseudocode.



Test Surface - Black



Test Surface - Gray



Test Surface - White





Python with Robots Mission 5 Lab Data Sheet – OBJ 2		Name:	
Obj. 2 Line Sensors: What reading does a line sensor give for different materials?			
Lighting (room, light, dark, etc.)	Background (carpet, tile, etc.)	Reflective surface (black tape, paper, etc.)	Line sensor reading Value between 0-4095
Planning Guide: Use this paper to plan your program with a flowchart or pseudocode.			



Python with Robots Mission 5 Lab Data Sheet – OBJ 2	Name:
Obj. 2 The Debug Console: Distance - How does the distance from the surface impact the line sensor read values? <i>Remember your constants - you must use the same surface throughout the test!</i>	
Distance (in cm)	Analog Value (range 0-4095)
0.5 (distance from surface when sitting flat)	

Surface - How does the reflectivity of the surface impact the line sensor read values? <i>Remember your constants - you must use the same distance throughout the test!</i>	
Surface description	Analog Value (range 0-4095)



**Python with Robots
Mission 6 Lab Data Sheet – OBJ. 2**

Name:

Obj. 2 Essential Question - How can we use REPL to interact with CodeBot's Python environment?

Complete the REPL practice below.

Command	Return
check_lines(2500)	
ls.read(0)	
ls.read(1)	
ls.read(2)	
ls.read(3)	
ls.read(4)	

Now experiment with your own commands below!

Command	Return



Python with Robots Mission 6 Lab Data Sheet – OBJ 3, 4, 5	Name:
--	--------------

Obj 3 (Line sensing): When will the sensors detect TRUE for a given threshold and reflecting surface? Use REPL and the Console Panel. Try different thresholds and reflective surfaces. Write down the results.

Threshold (start with 2500)	Reflective surface	Sensor reading (true or false)

Obj 4 (Get a new threshold): For Objective #4, you need to determine a new threshold for the sensors. Using REPL and the Console Panel, start with `ls.check(0)`. See the range of raw values for your line/ground area. Try different threshold values until you find the one that works best with `ls.check()`.

Start with <code>ls.check(0)</code> What is the range of raw values:	
Now choose different values as a threshold :	Result:

Obj 5 (Between the edges): For Objective #5, you want to experiment with different curves and turns.

Speed of 'bot	Turn code (L / R)	Type of Curve	Result:



Python with Robots
Mission 6 Lab Data Sheet – OBJ 5

Name:

Obj 5 – Line Follower Reflection

Essential Question: What limitations does the LineFollow1 program have? Test your line follower and answer the questions below.

What situations does this algorithm handle nicely?

What situations make it *fail*?

Why does it fail in those cases?

Collected Data

Line Position	LEDs(vals)
Far Left	
Center	
Far Right	



**Python with Robots
Mission 7 Lab Data Sheet – OBJ 1**

Name:

Objective 1: Presence Detector

Essential Question - What are “ideal” conditions in which to use proximity sensors? How can we program the ‘bot to adapt to its environment?

Purpose: The purpose of this lab is to determine ideal values for a given surface.

Procedure:

1. Partner A will run the ‘bot and program; Partner B will measure and record data.
2. Complete the lesson in CodeSpace and stop where it says to “*experiment with the code*”
3. Ensure you are keeping **all** conditions the same except for the variable you are testing.
4. With the program running, Partner A will line up the front edge of the ‘bot on the test surface.
5. Partner B will line up the edge of the meter stick at the same front edge. Keeping your body out of the way of the sensors, place your white card outside its range and slowly bring it toward the ‘bot. Note when the proximity sensors detect the object. **Note-You may find that one sensor is more sensitive than the other. Be careful to be consistent in your procedure.*
6. Repeat 3 times to ensure you are getting similar results.
7. Partner B will note measurements in the data table.
8. Repeat for each condition

Materials:

- CodeBot, CodeSpace, USB cable
- Test surfaces (white, gray, black)
- Meter stick
- White notecard or paper

Hypothesis:

What are the constants in your experiment?



Python with Robots Mission 7 Lab Data Sheet – OBJ 1	Name:
Obj. 1 Presence Detector: Use the “Test Surface” color blocks to find the distance needed for the proximity sensor to detect the surface color. You can also use other surfaces.	
Color of surface	Distance for True
Open space	
Black	
Gray	
White	



Python with Robots Mission 7 Lab Data Sheet – OBJ 2, 4, 7	Name:
--	--------------

Obj. 2 Power and Threshold: What power and threshold work best for an environment? Select one object. Then use different power and threshold levels on different surfaces and see which ones work best.

Object / Surface	Power setting	Threshold setting	Result

Obj. 4 Distance: After calibrating your ‘bot, you will want to see if the threshold chosen is working. Watch the debug console to see what thresh values it picks up. Try different object distances to make sure it is working.

Object / Surface	Distance	Threshold setting	Result

Obj. 7 Power: Define a function that calibrates the power of the flashlight. Then test the calibration on different surfaces.

Object / Surface	Power setting	Threshold setting	Result



Python with Robots Mission 7 Lab Data Sheet OBJ 2 – Power	Name:
--	--------------

Testing Power

An emitter power level setting from 1 (low power) to 8 (high power) controls the brightness of CodeBot’s IR “flashlight.”

power = 1
 thresh = 75

Color of surface	Distance (cm) for True
Open space	
Black	
Gray	
White	

power = 4
 thresh = 75

Color of surface	Distance (cm) for True
Open space	
Black	
Gray	
White	

power = 8
 thresh = 75

Color of surface	Distance (cm) for True
Open space	
Black	
Gray	
White	



**Python with Robots
Mission 7 Lab Data Sheet
OBJ 2 – Threshold**

Name:

Testing Threshold

A detection threshold from 0%-100% controls how much light is needed for a True detection.

power = 4
thresh = 25

Color of surface	Distance (cm) for True
Open space	
Black	
Gray	
White	

power = 4
thresh = 50

Color of surface	Distance (cm) for True
Open space	
Black	
Gray	
White	

power = 4
thresh = 100

Color of surface	Distance (cm) for True
Open space	
Black	
Gray	
White	



Python with Robots
Mission 7 Lab Data Sheet – Reflection

Name:

Discussion:

1. Was there anything inconsistent with the way you carried out your experiment? If so, what was it?

2. What environmental factors besides the color of the surface could affect the sensor readings?

Conclusion:

My hypothesis was supported / not supported by the experiment. How do you know?

What are your next steps? How could you share your knowledge with others?



Python with Robots Mission 8 Lab Data Sheet	Name:
--	--------------

Obj. 9 Speedometer Part 1: Follow these steps:

1. Change the distance traveled to a constant 10 cm: `drive(10)`
2. Change the power value to a constant 50%
`motors.run(LEFT, 50)`
`motors.run(RIGHT, 50)`
3. Run the program **10 times** and find the average of all 10 print values and record below.
4. Change the surface and repeat.

Floor Surface	[LEFT] speeds	AVG	[RIGHT] speeds	AVG

Obj. 9 Speedometer Part 2: Follow these steps:

1. Keep the distance traveled to a constant 10 cm: `drive(10)`
2. Choose ONE surface for the experiment and use it as a constant.
3. Use the same power for each wheel, but change the power incrementally.
`motors.run(LEFT, 10)`
`motors.run(RIGHT, 10)`
4. Run the program **10 times** and find the average of all 10 print values and record below. Change the power values by 10 each time.

Power	[LEFT] speeds	AVG	[RIGHT] speeds	AVG
10				
20				
30				
40				
50				
60				
70				
80				
90				
100				



Python with Robots Mission 8 Lab Data Sheet – Surface Test	Name:
<pre> from botcore import * from time import sleep motors.enable(True) motors.run(LEFT,50) motors.run(RIGHT,50) sleep(1.0) motors.enable(False) </pre>	
Surface	Distance traveled (cm)
Wood	
Vinyl	
Tile	
Carpet	
<i>other</i>	



Python with Robots Mission 8 Lab Data Sheet	Name:
Wheel Encoder test	
Open slot	Closed slot (spoke)
Average:	Average:



Python with Robots Mission 9 Lab Data Sheet – OBJ 3, 6	Name:
---	--------------

Obj. 3 Battery Tester with Load: Follow these steps:

1. Start a new file and create a test program as shown ->
2. Start with no LEDs turned on. Run the code.
3. Check the REPL console.
4. Record the battery voltage and percentage.
5. Repeat, lighting another LED each time.

```

from codebot import *
from time import sleep_ms

leds.user(0b00000000)
v = system.pwr_volts
pct = (v / 2) - 2
print(v, pct)
sleep_ms(50)
# turn off LEDs
leds.user(0b00000000)
        
```

LEDs turned on	Voltage	Percentage
0		
1		
2		
3		
4		
5		
6		
7		
8		

Obj. 6 Temperature Check: Follow these steps:

1. Use your code from Objective 6
2. Start with sleep_ms(200) and record the average temperature
3. Repeat the experiment, changing the amount of delay. Try faster times and slower times.
4. Record the average temperature for each delay.

Amount of delay	Average temperature	
200		



Python with Robots
Mission 9 Lab Data Sheet – Battery Test

Name:

```
from botcore import*
from time import sleep

def vbatt_load():
    #Read battery voltage under load with user LEDs on/off
    leds.user(0b00000000)
    v = system.pwr_volts()
    leds.user(0)

    return v
```

Number of leds.user on

My battery capacity %

0	
1	
2	
3	
4	
5	
6	
7	
8	



Python with Robots Mission 9 Lab Data Sheet – Temperature	Name:
--	--------------

```

from botcore import*
from time import sleep

while True:
    temp = system.temp_F()
    samples.append(temp)
    if len(samples) == 5:
        average = avg_list(samples)
        samples.clear()
        print ("Average temp: ", average)
    
```

Time	Average Temperature (F/C)
0.0 sec	
0.5 sec	
1.0 sec	
1.5 sec	
2.0 sec	
2.5 sec	
3.0 sec	
3.5 sec	
4.0 sec	
4.5 sec	
5.0 sec	