FIRIA LABS

Curriculum Guide

Clear all *Checkpoints* in 3 minutes or less

Objective 7: Proportional Control

Mission Pack:

Level-1 Python with Virtual Robotics

Table of Contents

Level-1 Python with Virtual Robotics Overview

Designed as an advanced Computer Science elective for students in grades 10-12, this course introduces the fundamentals of Python programming through immersive virtual robotics simulations. Students will write code to control their virtual robot and see immediate results, enhancing their understanding of programming concepts in a realistic, engaging environment. Featuring virtual sensors and actuators modeled on real components, the skills learned are directly transferable to physical CodeBots. Through challenging missions, students explore key computer science concepts and engage in meaningful, interactive learning. This curriculum not only prepares students for Python certification but also bridges the gap between virtual and real-world robotics.

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.firialabs.com/curricula/cs-unplugged.

Mission 1: Welcome

Take a tour of the CodeSpace Development Environment. Learn how to use the text editor, hints, CodeTrek and tools.

Mission 2: Introducing CodeBot

Get to know your friendly neighborhood Virtual Robot! Locate the motors, LEDs and buttons on the ‘bot.

Mission 3: Light the Way

Light up those LEDs and get CodeBot flashing. Learn about bits and binary while creating a light show.

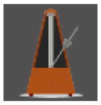
Mission 4: Get Moving

Get your motors running… Head out on the Virtual Highway! Use the motors to move the ‘bot around the 3D environment, and play notes along the way.

Mission 5: Dance Bot

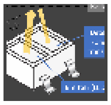
Does CodeBot have what it takes to win a dance competition? Combine movement with sweeping LEDs to dance CodeBot around the floor. Use loops and functions and a button press.

Mission 6: Robot Metronome



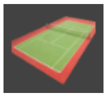
Write code to make a time-keeping Python Maestro! Learn about dictionaries and files while turning CodeBot into a jukebox.

Mission 7: Line Sensors



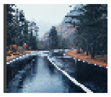
Use the line sensors to navigate your robot. Use a matrix to document a compass position and surface color to point the ‘bot in the right direction.

Mission 8: Boundary Patrol



Program your CodeBot to roam a fenced area, using line sensors. Use sensor data to make smart decisions on which way to turn.

Mission 9: Line Following

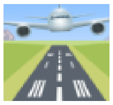


Tune up your Line Sensors and hit the road on the biggest line-course around. Use sensor data to make turns and accelerate autonomously.

Mission 10: Fido Fetch

Train your CodeBot to fetch using a dictionary of commands! Type input to the console and play with your robot dog.

Mission 11: Airfield Ops



Learn some unique programming concepts to help with airfield runway operations! Combine math operations with line sensor data for a useful application.

Mission 12: King of the Hill

Harness the CodeBot's accelerometer to climb to the top of a mountain! Unpack a tuple of sensor data to control the 'bot autonomously over rough terrain.

Mission 13: Going the Distance

Learn all the details of CodeBot's wheel encoders. Use the sensor data to drive a specified distance or maintain a specific speed.

Mission 14: Music Box

Turn the CodeBot into a jukebox while learning about Python file operations. Review CodeBot's speaker capabilities.

Mission 15: Cyber Storm

Help protect an email server by using file operations! Review file operations from the last mission, and learn some more in this useful application.

Unit 1: Introductory Missions (5-8 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 turn on and off CodeBot's LEDs while learning about bits and binary.

Summary of Mission 1:

Students start by becoming familiar with CodeSpace. They learn about the Mission Objectives, the text editor, CodeTrek, the Toolbox, and the camera simulation controls.

Summary of Mission 2:

Students learn about CodeBot and its peripherals. They identify parts of CodeBot by clicking on the item in the virtual environment.

Summary of Mission 3:

Students learn basic Python code, like importing a module and turning on an LED. They learn about delaying code using a sleep() function and using binary patterns to control the LEDs.

Summary of Mission 4:

Students get the motors running on CodeBot and play short tones using the speaker. They learn to move the 'bot in a spin, and also move forward with varying speeds.

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.

Standards addressed in this unit:

CSTA Standards Grades 9-10	CSTA Standards Grades 11-12
• 3A-CS-03 • 3A-DA-09 • 3A-AP-16 • 3A-AP-19 • 3A-AP-21 • 3-AP-22 • 3-AP-23	• 3B-CS-02 • 3B-AP-17 • 3B-AP-22

Mission 1: Welcome

Time Frame: ½ hour

Project Goal: Students will learn about the CodeSpace learning environment.

Learning Targets

- I can navigate CodeSpace.
- Identify major parts of the Codespace interface: Mission Bar, Objective Panel, text editor, CodeTrek, Toolbox, and Lesson Navigation Controls

Key Concepts

- 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.

Assessment Opportunities

- Adding a tool to the toolbox (Obj. 1.3)
- Quiz after Objective 4
- Print a picture of CodeSpace and have students label the parts (teacher resources)

Success Criteria

☐ Navigate CodeSpace

☐ Identify major features of the CodeSpace interface: Editor panel, Lesson panel, Toolbox, CodeTrek, Hints

Vocabulary

- **Objective:** The steps in the mission; has a goal to accomplish
- **Text editor:** Where you type the code
- **Code:** Instructions to the computer
- **Debugging:** The process of understanding what the computer is actually doing and then changing the code to do what you want it to do
- **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

New Python Code

Real World Applications

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.

Teacher Notes:

- 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.
- A worksheet with a picture of CodeSpace to label is available at learn.firialabs.com.

Extensions / Cross-Curricular

Mission 2: Introducing CodeBot

Time Frame: ½ hour

Project Goal: Students will learn about the peripherals of CodeBot.

Learning Targets

- I can identify the main components of the CodeBot.
- I can state the purpose of a peripheral as input or output.

Key Concepts

- There are a lot of hardware peripherals on the CodeBot, including sensors, LEDs, motors, buttons, and a speaker.
- The camera controls can be changed in the virtual environment.

Assessment Opportunities

- Print a picture of the CodeBot and have students label the parts. (teacher resources)
- Print a picture of the CodeBot and have students label the inputs and outputs.

Success Criteria

- ☐ Identify the parts of the CodeBot.
- ☐ Identify the purpose of the parts (input or output).

Vocabulary

- **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

New Python Code

Real World Applications

Discuss the fact that all the electronic devices they use have circuit boards similar to CodeBot 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!

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.

Teacher Notes

- A worksheet with a picture of CodeBot to label is available at learn.firialabs.com.
- Review "input" and "output". For each peripheral, discuss if it is used for input or output.
- 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.

Mission 3: Light the Way

Time Frame: 1 hour

Project Goal: Students will learn the basics of Python

Learning Targets

- I can import a library.
- I can use a predefined function from a library.
- I can use comments to explain my code.
- I can turn on a single user LED.
- I can use binary to turn on multiple LEDs.
- I can use decimal values to turn on or off LEDs.
- I can delay program execution using the sleep() function from the time library.

Key Concepts

- Adding comments and blank lines in your code makes it easier to read.
- Built-in functions come from libraries, like botcore or time.
- The CodeBot has 17 LEDs, and each can be turned on or off using code.
- Use the prefix 0b in front of a binary pattern to control LEDs.
- Use the sleep() function to slow down the computer.

Assessment Opportunities

- Quiz after Objective 3
- Submit program code, or give students printed code and have them explain each line:
    - Objective 3, Objective 4, Objective 5
- Short quiz or exit ticket that demonstrates the binary code for turning on a specific pattern of user LEDs (similar to Objective 3)

Success Criteria

□ Create a new file for program code.
□ Turn on a single user LED.
□ Control all user LEDs using binary.
□ Control all user LEDs using decimal.
□ Use the sleep() function to animate the LEDs.
□ Animate CodeBot's LEDs in a controlled sequence.

Vocabulary

- **API:** Application Programming Interface – details how your program interacts with different services.
- **Library:** A module that contains pre-built functions and code that can be referenced after the library is imported.
- **Byte:** 8 bits, which are binary digits.
- **Binary:** Base 2 (digits are 0 and 1, or off and on).
- **Bit:** Binary digit.
- **Comments:** Code that doesn't get run; notes in the code about what you are doing.
- **Shift register:** An electronic circuit that allows an array of output pins to be set HIGH or LOW based on a sequence of binary digits that are shifted into a single pin.
- **Animation:** A sequence of changes, at a controlled speed.
- **Delay:** Functions that slow things down, like sleep(); the module must be imported first

New Python Code

`from botcore import *`	Import the botcore library module to access its hardware and pre-built functions
`leds.user_num(0, True)`	Turn on one user LED (LED numbers 0-7, True=on, False=off)
`leds.ls_num(0, True)`	Turn on a line sensor LED (LED numbers 0-4, True/False)
`# Turn on LEDs 0, 3, 4, 7`	Comment

`leds.user(0b11111111)`
`leds.ls(0b11111)`
	Turn on LEDs using binary. User LEDs have 8 bits.
	Line sensor LEDs have 5 bits. Use the prefix 0b to indicate binary.

`from time import sleep`	Import the sleep function from the time library module
`sleep(0.2)`	Delay program execution by seconds
`leds.user(0)`	Turn off user LEDs using decimal

Real World Applications

You've used a fundamental computer science and robotics principle. Similar code is used in cars, stage lights, video games, and more!

- Controlling LEDs with specific timing and sequencing

Teacher Notes

- It is not explicitly mentioned, but students should start a new file for this program code. File names should be descriptive. If you are having students submit the file, you may want them to include their name in the filename.
- Discuss readability in code, such as using comments and adding blank lines.
- Students don't need to know how to convert decimal and binary numbers, but it is a fun topic. You can spend extra time on binary numbers as an option.
- Give students time to experiment with controlling the LEDs with binary and with decimal.

Extensions / Cross-Curricular

- **CHALLENGE:** Blink all the LEDs on and off in an animation pattern. Repeat the animation at least twice.
- **CHALLENGE:** Use only binary to animate the LEDs.
- **CHALLENGE:** Use only decimal numbers to animate the LEDs.
- **SCIENCE:** Have a lesson on LEDs and light.
- **MATH:** Develop an algorithm to decide how long to turn on or off the lights for a sequence that lasts a set amount of time.

Mission 4: Get Moving

Time Frame: 1 hour

Project Goal: Students will move CodeBot by enabling and giving power to the wheels.

Learning Targets

- I can apply power to the wheels to spin the CodeBot.
- I can apply power to the wheels to move the CodeBot in a curve (or circle).
- I can control the movement of CodeBot to tag objects.
- I can play tones on CodeBot's speaker.

Key Concepts

- Motors must be enabled before they can receive power.
- Wheel power ranges from -100 percent (reverse) to +100 percent (forward) speeds.
- If equal but opposite speeds are applied to the wheels, the 'bot will spin.
- If positive but different speeds are applied to the wheels, the 'bot will move in a circle.
- Use the sleep() function to keep the motors running.
- The speaker can play a tone for a specified amount of time.

Assessment Opportunities

- Quiz after Objective 3
- Submit program code, or give students printed code and have them explain each line:
  - Objective 1, Objective 2, Objective 3, Objective 4
- Code Tracing Chart
- Quiz or exit ticket with code. Have students explain the direction the 'bot will move.
- Objective 3 Planning Guide
- Submit final program code for robot tag
- Submit Obj. 4 code for "Sound Off"

Success Criteria

☐ Use wheel power to spin CodeBot
☐ Use wheel power to move CodeBot in a circle
☐ Use wheel power to move CodeBot so it tags tennis balls
☐ Play at least two notes from the speaker

Vocabulary

New Python Code

`motors.enable(True)`	Enable CodeBot's motors

`motors.run(LEFT, 30)`
`motors.run(RIGHT, -30)`
	Apply power on the wheels to spin.
	(Positive left, negative right will spin clockwise)
	(Negative left, positive right spin counter clockwise)

`motors.run(LEFT, 50)`
`motors.run(RIGHT, 40)`
	Sweeping circle (or curve) in a clockwise direction

`spkr.pitch(440) / sleep(0.1)`	Play a tone on the speaker for a specific amount of time
`spkr.off()`	Turn off the tone

Real World Applications

Using motors, lights and sounds in a timed sequence pattern is animation. Using code for the animation makes it automation! Many items and objects in the real world use automation.

- Amusement park rides, games, cars, stage lights, automatic vacuums, etc.
-

Have students discuss where they see animation in their lives, and the code that may be running it.

Teacher Notes

- It is not explicitly mentioned, but students should start a new file for this program code. File names should be descriptive. If you are having students submit the file, you may want them to include their name in the filename.
- Students can use the same file during the mission and change the code, or start a new file each time.
- A document for planning their code for Objective 3 is available at learn.firialabs.com.
- Students may want to zoom out or change the camera view when running the code to see the 'bot move.
- Give students time to experiment with different speeds. No need to rush through the objectives.
- Give students time to experiment with different tones. No need to rush through the objectives.
- Do a trace chart together. Talk about what works and what doesn't. Emphasize the importance of documenting your tries to help with debugging in the future.

Extensions / Cross-Curricular

- For Objective 3, tag all four tennis balls.
- Play a short song, like a nursery rhyme, using the speaker.
- **LANGUAGE ARTS:** Have students write a short review about an animation or automation.
- **PERFORMING ARTS:** Learn about music and note notation.

Unit 1 Remix Project and Exam	**Time Frame:** 2-3 hours
Remix Project Goal: Students will use the skills and concepts they learned in the first four missions to create their own project. **Remix Project Outline:** Follow the five-steps of the design process (document on next page) to design a remix project.	**Exam Goal:** Students will show mastery of Python and robotics concepts by obtaining at least 75% correct on a multiple choice test. **Project Outline:** Review concepts through additional practice and Kahoots. Then students take the exam. An option for Microsoft Forms is available. Vocabulary and concepts are two different tests.
Remix Project Assessment Opportunities ● Unit 1 Remix Project Planning Guide ● Peer reviews / Gallery walk ● Unit 1 Remix Project Rubric Checklist ● Submit Unit 1 Remix Project Program	**Unit 1 Exam Opportunities** ● 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)

Supplementary Materials (available at learn.firialabs.com)

- Unit 1 Remix Project Planning Guide
- Unit 1 review and test questions
- Code Tracing Chart

Unit 1 Remix Project Ideas:

- Combine extension ideas from Mission 3 or Mission 4.
- Use the binary pattern for turning on LEDs, and combine that with ‘bot movement and sound.
- Move the ‘bot in a specific pattern, like a square or diamond. While moving, flash lights and/or play notes.
- Think of your own creative project with movement and LEDs and sound.

Unit 1 Remix Project Rubric Checklist:

☐ New file is used and filename is descriptive
☐ Moves the CodeBot forward and/or backward one or more times
☐ Turns the CodeBot one or more times
☐ Uses a sleep delay one or more times
☐ Turns on one or more LED lights
☐ Plays at least two notes using the speaker
☐ Includes comments and whitespace for readability
☐ Code runs with no errors

Unit 2: Inputs and Outputs (6-12 hours)

Students continue their programming journey by learning about and using several programming concepts and skills. Techniques introduced include the while loop, for loop, if statement, infinite loop and functions. Students will use Boolean variables, logical operators and lists. Variables are used extensively, including initializing, assigning and incrementing.

Summary of Mission 5:

This Mission is full of programming concepts and skills. Combine movement with sweeping LEDs to make CodeBot dance on the auditorium floor. Use loops and functions to reduce code repetition. Control the ‘bot by using a button press.

Summary of Mission 6:

This mission is also full of programming concepts and skills. Students learn about Boolean variables, if statements for branching, lists, accessing an item in a list, and states of a program. The final program uses an infinite loop to change CodeBot into an interactive metronome. Using a Boolean variable, the sound can be toggled on and off. Using a list, incrementing an index, and wrap-around, the tempo of the metronome can change between five different tempos. LEDs indicate tempo and sound.

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)

Standards addressed in this unit:

CSTA Standards Grades 9-10	CSTA Standards Grades 11-12
• 3A-CS-01 • 3A-CS-03 • 3A-DA-11 • 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	• 3B-CS-02 • 3B-AP-10 • 3B-AP-12 • 3B-AP-14 • 3B-AP-16 • 3B-AP-17 • 3B-AP-20 • 3B-AP-22 • 3B-AP-23

Mission 5: Dance Bot

Time Frame: 2-4 hours

Project Goal: Students will use functions and loops to create a dancing CodeBot.

Learning Targets

- I can create a new file for code.
- I can use a while loop to flash the user LEDs.
- I can use the debugger to step through a program and track variables.
- I can print a message to the console.
- I can use a for loop to flash the user LEDs.
- I can use a loop variable to control a user LED.
- I can use a for loop with optional arguments to sweep LEDs backwards.
- I can define and use functions for sweeping LEDs.
- I can use a button to control the 'bot.
- I can move the CodeBot using the sweeping LEDs as the time element.

Key Concepts

- Computers execute code in sequential steps.
- The CodeSpace debugger lets you *step* through the code one line at a time to understand what the computer is doing.
- Loops allow a block of code to be repeated without typing it again. Python has two types of loops: the while loop and the for loop.
- 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.
- Increments (and decrements) are used for updating variables like counters.
- A function is a named chunk of code you can run anytime by calling its name.
- Button presses (inputs), LEDs (outputs) and speaker sounds (outputs) are part of the user interface. They allow the user to interact with the CodeBot.

Assessment Opportunities

- Quiz after Objective 2
- Quiz after Objective 6
- Submit program code, or give students printed code and have them explain each line:
    - Objective 1
    - Objective 5
- Code Tracing Chart for any objective
- Return to an earlier mission and have students define functions for repeated code
- Exit ticket: Explain the difference between the two ways to check the state of a button
- Submit final “dancebot.py” program

Success Criteria

- □ Use a while loop to flash user LEDs
- □ Use a for loop and loop variable to flash user LEDs
- □ Use two for loops to sweep across the LEDs
- □ Define functions for sweeping left and right
- □ Call the functions in the main program
- □ Check the state of a button press
- □ Use a while loop to sweep the LEDs until button 0 is pressed
- □ Move the ‘bot around the surface and touch at least 5 balloons while sweeping the LEDs

Vocabulary

- **Sequential:** Running code in order (sequence) one line at a time.
- **Loop:** Repeating a section of code, subject to a condition.
- **Indenting:** A way to structure blocks of code by offsetting a block of code four spaces; blocks of code are indented following a statement with a colon (:).
- **While loop:** A while loop repeats a block of indented code while a condition is true.
- **Variable:** A label, or name, for a container that stores information for a computer to use.
- **Initialize:** The first, or initial, value assigned, or stored, in a variable.
- **Increment:** Updating a variable by adding 1 to it (like counting).
- **Bug:** A problem with the code where it doesn’t do what you expect it to do.
- **Debugger:** An advanced tool in CodeSpace that gives more control over a program being debugged. The buttons are available in the debugger: step next, step into and step out. Also, variables are tracked under “Locals” and “Globals”.
- **String formatting:** You can change the formatting of printed text using key arguments.
    - end=’,’ will add a comma after each item printed.
    - end= ‘ ‘ will add a space after each item printed.
- **For loop:** Repeating a section of code across a range of numbers or items in a list. It automatically initializes and updates the loop variable.

- **Readability:** Code that is easy to understand, like using blank lines to separate sections of code.
- **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.
- **Function:** A named chunk of code you can run anytime just by calling its name; a way to reuse code without retyping it. The code in a function doesn't run until you call it.
- **Editor shortcuts:** Keyboard hotkeys to write code faster; combinations of keys to complete a task.
- **Iterating:** Moving through a sequence, like in a for loop or list, one item at a time.
- **Parameter:** A variable defined as part of a function definition that receives an argument (value) from the function call.

New Python Code

`count = 0`
`while count < 0:`
`    leds.user_num(0, True)`
`    sleep(0.1)`
`    leds.user_num(0, False)`
`    sleep(0.1)`
`    count = count + 1`
	While loop for repeating code while a condition is true.

`count = 0`
`count = count + 1`
	Assign a value to a variable.
	(use the assignment operator =)

`count = 0`	Initialize a variable.
`count = count + 1`	Increment (or update) a variable.

`print(count)`
`print(count, end=‘ ’)`
	Display a message on the console. The panel must be open to see the message.

`for count in range(8):`
`    print(count)`
`    leds.user_num(count, True)`
`    sleep(0.1)`
`    leds.user_num(count, False)`
`    sleep(0.1)`
	For loop with range() function. This example also uses the loop variable (count) as part of an instruction.

`for count in range(6, -1, -1):`
`    print(count)`
`    leds.user_num(count, True)`
`    sleep(0.1)`
`    leds.user_num(count, False)`
`    sleep(0.1)`
	For loop with full range() function and optional arguments to count backwards.

`def sweep_left():`
`    for count in range(8):`
`        print(count)`
`        leds.user_num(count, True)`
`        sleep(0.1)`
`        leds.user_num(count, False)`
	Define a function. The indented code following the colon (:) is executed when the function is called.

`sweep_left()`
`sweep_right()`
	Function calls. Notice, no colon(:), but the call does include parenthesis (). The function call is not indented.

`buttons.is_pressed(0)`	Checks the current state of a button press.
`buttons.was_pressed(0)`	Checks the last state of a button press.

`not buttons.was_pressed(0)`	Returns the opposite. For example, if buttons.was_pressed(0) is True, the statement will return False.

Real World Applications

Abstraction is used in daily life and in many applications. Abstraction is a key concept of computer science. Functions are a form of abstraction because they can hide the details of how a task is accomplished, which enables a problem to be simplified and to focus on the parts that need attention.

- Look up definitions of abstraction, and then come up with your own definition.
- Look for examples of abstraction in everyday life. One example is how a car works. You don't need to know how the engine works, or how the gas pedal makes the car go faster to drive a car. What other examples can you discuss?

While loops, for loops and functions are frequently used in animation and automation.

- Discuss animations and/or automations in the real world that have repetitive tasks that can be simplified using loops.

Algorithms are a part of everyday life. They are step-by-step instructions to complete a task.

- Think about daily activities that you do. What is the algorithm for the task?

Teacher Notes

- Functions are a form of abstraction. This key concept isn't explicitly in the instructions, but this mission is a good place to start the conversation about abstraction and how it is used in a program. See "Real World Applications" for some ideas.
- There is a lot of information in this Mission. Take your time on each objective. Give extra practice or review as needed.
- 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.
- Use the assessment opportunities to check for understanding.
- Have students fill out a Code Tracing Chart during one of the objectives and discuss it with them. Have a class discussion.
- The last objective can be frustrating for students because the balloons and the 'bot don't perform the same way each time. This could be a good time for students to work in pairs and problem solve together. Also, remind students to try the same code more than once to see what happens. On any given program execution, the code may reach the goals.

Extension / Cross-Curricular

- Have a competition to see which 'bot can touch the most balloons.
- Use a loop in the main program to repeat calls to the go() function and sweep() functions.
- Use the sandbox and synchronize dance moves to music. Or after the goals are met, use the same dance floor to synchronize dance moves while moving balloons.
- **LANGUAGE ARTS:** Have students write the algorithm to their dance as numbered instructions.
- **SCIENCE:** Write the algorithm to the scientific method.
- **MATH:** Write the algorithm to solve an equation.
- **PHYSICAL SCIENCE:** Do experiment with wheel speed. Use the sandbox and classroom environment. Try different speeds and note the distance traveled by tile squares. Graph the results and then determine the rate using the distance = rate * time equation.

Mission 6: Robot Metronome

Time Frame: 2-4 hours

Project Goal: Students create an interactive metronome with LED indicators, sound and tempo selection.

Learning Targets

- I can create a simple metronome by flashing all user LEDs and playing a tone, once per second.
- I can update a variable using a mathematical equation.
- I can use an if statement to control program flow.
- I can use the logical operator 'not' to toggle a Boolean variable.
- I can use a bit-shift operator to control LEDs.
- I can define and use a list of tempos.
- I can wrap-around an index variable to start over when the length of the list is reached.

Key Concepts

- Infinite while loops are used to execute an algorithm continuously.
- Iterating with a while loop or a for loop reduces the need for repeated code.
- Both loops are versatile and can iterate forward or backward.
- Branching with an **if:** statement controls the flow of the program.
- A Boolean variable can be used as a toggle, or on/off switch.
- A bit shift operator is used to change which LED lights up.
- A list can hold multiple values.
- Each item in a list is in order and is accessed using an index.
- The state of the button press is True or False and can be used to control the CodeBot.

Assessment Opportunities

- Quiz after Objective 4
- Quiz after Objective 8
- Submit program code, or give students printed code and have them explain each line:
  - Objective 4, Objective 7
- Give extra practice with logical operators
- Give extra practice with lists
- Use the Code Tracing Chart for an objective
- Submit final "metronome.py" program

Success Criteria

□ Use a while loop to flash LEDs

□ Use two for loops to flash LEDs

□ Use an infinite loop to flash LEDs continuously

□ Toggle a Boolean variable to control sound and the power LED

□ Use a list for five tempos

□ Wrap around the list for continuous selection

Vocabulary

- **Intervals:** Time related functions available in Python to track time. A common function for a delay is sleep(). Common functions for checking the current elapsed time are time.time() and time.ticks().
- **Infinite loop:** A loop that doesn't end because the loop condition is always true.
- **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.
- **State:** The status of a system with transitions. Your program can only be in one of a known set of states at any given time.
- **Transition:** Moving between states. A program can transition from one state to another when certain conditions are met.
- **Boolean data type:** A data type that has two values: True or False.
- **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.
- **Banching (control flow):** Decision points in code; code will take a different branch or path depending on a condition.
- **Data type:** The kind of value stored in a variable. Common built-in data types are str (string), int (integer), float (decimal number), and bool (Boolean).
- **List data type:** A sequence of items that you can access with an index.
- **Index:** The position of an item in a list. The first index of a list is 0.
- **Logical operators:** Operators that compare multiple conditions: 'or' and 'and'.
- **Magic number:** A literal value used in a code.
- **Single equals =:** Assignment – used to assign a value to a variable.
- **Double equals ==:** Comparison operator to determine if two objects are the same.

New Python Code

`while True:`
`    leds.user(0b11111111)`
`    spkr.pitch(784)`
`    sleep(0.1)`
`    spkr.off()`
`    leds.user(0)`
`    sleep(0.9)`
	Infinite loop.
	Does not stop because the loop condition is always true. The code must be stopped manually by clicking the STOP button.

`pause = (60 / tempo) - beat_duration`	Assign a value to a variable
`sound_on = True`	Boolean variable

`if sound_on:`
`    spkr.pitch(784)`
`if buttons.was_pressed(0):`
`    sound_on = False`
	If statement (branching)
	Also known as control flow

`if buttons.was_pressed(0):`
`    sound_on = not sound_on`
	Toggle a Boolean operator
	(using the 'not' operator)

`leds.pwr(not sound_on)`	Use the opposite of a Boolean to turn on/off
`leds.ls(1 << tempo_select)`	Bit shift operator (shift bit to the left by index)
`tempo_list = [50, 70, 100, 140, 180]`	Define a list
`tempo = tempo_list[tempo_select]`	Access an item from a list using a variable for the index
`len(tempo_list)`	len() function – the number of items in a list

`if tempo_select == len(tempo_list):`
`    tempo_select = 0`
	Wrap-around an index variable when the length of the list is reached

Real World Applications

The concepts in the lesson are used in real world applications all the time: continuous looping, LEDs as indicators, lists, toggle for on/off. Challenge students to come up with examples of each.

Teacher Notes

- The first objective uses a binary pattern to turn on all user LEDs. Review from Mission 3.
- The second objective uses the speaker and tones. Review from Mission 4 as needed.
- When an infinite loop is used, the program will not stop automatically and must be stopped manually by pressing the STOP button.
- Review the difference between = and ==.
- There is a lot going on during this mission! Take your time and don't rush through the objectives. Create worksheets to review throughout the objectives. Some suggestions are given in assessment opportunities.

Extensions / Cross-Curricular

- Write the algorithm for the program.
- Add a function to the program.
- Add more tempos. Only 5 can show on the line sensor LEDs, so come up with a solution.
- **LANGUAGE ARTS:** Have students compare and contrast the user LEDS with line sensor LEDs. Or compare and contrast a while loop with a for loop.
- **SCIENCE:** Do an experiment with the metronome. Which is easier to follow – sound only, lights only, or both. Come up with your own experiment.
- **MATH:** The Mission shows how to assign a variable a value from an equation. Write a program with different equations and get solutions. Then make a graph with the data.

Unit 2 Remix Project and Exam	**Time Frame:** 2-5 hours
Remix Project Goal: Students will use the skills and concepts they learned in Mission 5 and Mission 6 to create their own project. **Remix Project Outline:** Follow the five-steps of the design process (document on next page) to design a remix project.	**Exam Goal:** Students will show mastery of Python and robotics concepts by obtaining at least 75% correct on a multiple choice test. **Project Outline:** Review concepts through additional practice and Kahoots. Then students take the exam. An option for Microsoft Forms is available. Vocabulary and concepts are two different tests.
Remix Project Assessment Opportunities ● Unit 2 Remix Project Planning Guide ● Peer reviews / Gallery walk ● Unit Remix Project Rubric Checklist ● Submit Unit 2 Remix Project Program	**Unit 2 Exam Opportunities** ● 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)

Supplementary Materials (available at learn.firialabs.com)

- Unit 2 Remix Project Planning Guide
- Unit 2 review and test questions
- Code Tracing Chart

Unit 2 Remix Project Ideas:

- Pick an extension idea from Mission 5 or Mission 6.
- Revisit an earlier Mission, like “Get Moving” and add functions, a list, and loops to the program.
- Make a dancing metronome and add movement to the program from Mission 6.
- Add synchronized sound and a list of movements to the dance bot program. Use a button to start the ‘bot dancing, and a button to go from different dance moves.
- Think of your own creative project with movement and LEDs and button input.

Unit 2 Remix Project Rubric Checklist:

□ Filename is descriptive
□ Uses at least one while or for loop
□ Uses an infinite loop for continuous execution
□ Turns on and off at least one LED
□ Plays a sound
□ Uses one or two buttons as input
□ Defines and calls at least one function
□ Uses at least one list
□ Uses at least one Boolean variable as a toggle
□ Code follows programming conventions of comments, readability, indenting, and capitalization
□ Code runs with no errors

Unit 3: Get Moving (9-18 hours)

Students learn how to optimize the CodeBot sensors and motors. All three missions use the line sensors to complete a task. Students learn and use several programming concepts, like multiple branching, lists and 2-dimensional arrays, dictionaries and functions. A button press and the input() function are used for user input.

Summary of Mission 7:

Students learn about line sensors and how to use the readings to control the CodeBot. The final program uses a 2-dimensional list, absolute values, and a return function. Students also get input from the user on the console by typing an answer.

Summary of Mission 8:

Students will use the line sensor readings to stay in bounds. The final program uses functions with default parameters, a list of Boolean values, and multiple branching for smarter decisions. Students learn that the speed of the 'bot and the direction of the turn make a difference in its responsiveness to its environment.

Summary of Mission 9:

Students use REPL and the console to try out Python code. Line sensors are used to stay on a black line and follow a complicated track. Dictionaries are introduced; how to define them and use them in code without getting an error. Additional functions are defined to control the 'bot for the fastest speeds while staying on the line.

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)

Standards addressed in this unit:

CSTA Standards Grades 9-10	CSTA Standards Grades 11-12
● 3A-CS-01 ● 3A-CS-03 ● 3A-DA-10 ● 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-22 ● 3A-AP-23 ● 3A-AP-26	● 3B-CS-02 ● 3B-DA-05 ● 3B-DA-06 ● 3B-AP-10 ● 3B-AP-12 ● 3B-AP-14 ● 3B-AP-15 ● 3B-AP-16 ● 3B-AP-17 ● 3B-AP-20 ● 3B-AP-21 ● 3B-AP-22 ● 3B-AP-23

Mission 7: Line Sensors

Time Frame: 2-4 hours

Project Goal: Students use sensor inputs to program the ‘bot to navigate around lines.

Learning Targets

- I can read a line sensor and print the value.
- I can print multiple numbers separated with a comma.
- I can use threshold comparison operators to make decisions with sensor data.
- I can break out of a loop.
- I can use the built-in abs() function.
- I can define and use a constant.
- I can create a 2-dimensional array.
- I can define a function that returns a value.
- I can program the CodeBot to respond to a specified color.

Key Concepts

- The data from an analog sensor needs to be changed to digital numbers for computing.
- The CodeBot has five line sensors, and each one gives its own brightness reading.
- Python has many built-in functions for math. Absolute value is one of them.
- A threshold comparison operator lets you check if a value is within a range.
- A user can input data using an input() function and typing an answer on the console.
- A matrix, or 2-dimensional array, can be used to store two pieces of data that are related, like a compass direction and brightness reading.
- A function can return a value to a variable.

Assessment Opportunities

- Quiz after Objective 1
- Quiz after Objective 3
- Quiz after Objective 6
- Obj. 2 Line Sensor Readings Chart
- Submit program code, or give students printed code and have them explain each line:
  - Objective 3, Objective 5
- Give extra practice with functions that return
- Give extra practice with 2-dimensional arrays
- Use the Code Tracing Chart for an objective
- Submit final “line_sense.py” program

Success Criteria

- ☐ Create a color detection program using line sensors that detect brightness.
- ☐ Display data on the console.
- ☐ Use abs() in a comparison.
- ☐ Break out of a loop if a condition is met.
- ☐ Define and use a constant.
- ☐ Define and use a matrix.
- ☐ Get input from the user on the console.
- ☐ Program the CodeBot to respond to a specific brightness level (color).

Vocabulary

- **Line sensors:** Photo reflective sensors that detect lines and boundaries beneath your ‘bot.
- **Analog:** Infinite variation, like from dark to light or cold to hot.
- **Int data type:** A value that is an integer; designated by int in Python; can be positive or negative.
- **ADC:** Analog to digital converter.
- **Constant:** A name, like a variable, that represents data in your code that doesn’t change.
- **Iterative process:** Repeatedly taking small steps to build a whole solution.
- **Matrix:** A list of lists; also known as a 2-dimensional array.

New Python Code

`left = ls.read(0)`
`right = ls.read(4)`
`reading = ls.read(num)`
	Read the brightness level detected by a line sensor

`print(left, right, sep=’,’)`	Print numbers separated by a comma
`if 3547 < left < 3567:`	Threshold comparison operator to check if a value is within a range
`break`	Break out of a loop
`motors.enable(False)`	Stop the motors
`MIN_DIFF = 100`	Define a constant (use all CAPS)

abs(left-right)	Find the absolute value

sensor_data = [
    [‘N’, 3557],
    [‘E’, 286],
    [‘S’, 1391],
    [‘W’, 2481]
]
	Define a matrix
	(a list of lists)

target_dir = input(“Enter direction: “)	Prompt user for input using the console
return d[0]	Function return
found = find_name(left)	Call a function with a return

Real World Applications

From robot housekeepers to self-driving cars, the sensing and control techniques used in this Mission apply to all kinds of intelligent systems. Analog sensors are non-contact sensors used in many industrial and commercial applications. Code like that used in this Mission impacts your life every day!

- Automatic guided vehicles use this kind of code to zoom around warehouse distribution centers.
- Robots are used to clean up environmental waste, explore underground mines, and discover shipwrecks in the deepest oceans.

Teacher Notes

- A document for Objective 2 Line Sensor Readings is available at learn.firialabs.com.
- During objective 4 an if statement is added to the code. The existing if statement is ‘nested’ inside the new if statement, so make sure students are indenting the code properly.
- A lot of code is added to Objective 6, and a lot of code is deleted. Students need to be very careful about their indenting, parenthesis, typing, spelling, etc.
- Remember to import the sleep() function from time for Obj. 6.
- For Obj. 6, zoom out to find the water bottle. Then open the console log to type in the correct direction when the program runs. Use a capital letter!
- A lot of new concepts are introduced very quickly. Take time to review as needed. You may even want to give extra practice.

Extensions / Cross-Curricular

- Add a button to stop the ‘bot.
- Add a beep when the correct color is found.
- Light up user LEDs to communicate what direction the CodeBot is facing. Flash the LEDs when the color is found.
- **LANGUAGE ARTS:** Have students write a summary of their project, using technical terms.
- **SCIENCE:** Have a lesson on how the line sensor works.
- **SCIENCE:** Have a lesson on brightness and reflectivity.
- **MATH:** Experiment with different thresholds and line sensors. Make a chart of the results, and graph the data.
- **MATH:** What if there wasn’t a built-in abs() function? What are some alternative solutions to getting a positive answer every time?

Mission 8: Boundary Patrol

Time Frame: 2-4 hours

Project Goal: Students will use sensor inputs to program the ‘bot to roam a fenced area.

Learning Targets

- 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 use a list of Boolean values to indicate if a threshold has been reached for each sensor.
- I can define a function that returns a list of Boolean values.
- I can teach the ‘bot to stay inside the lines.
- I can make a contact counter to show each line-detect on the user LEDs.
- I can use default parameters and multiple branching to make smarter decisions.

Key Concepts

- A threshold is a value in between two different sensor readings.
- Default parameters give functions flexibility.
- A list can be defined as empty, and new items can be appended to the end of the list.
- A list name is usually plural, since it can hold multiple items. Note the difference between val when a single sensor is read and vals when all sensors are read and stored in a list.
- A list can hold Boolean items. Each item can indicate if a line sensor is above or below a threshold (True or False).
- A list of Boolean items can control LEDs.
- An if:elif:else statement gives multiple branches.

Assessment Opportunities

- Quiz after Objective 2
- Quiz after Objective 4
- Submit program code, or give students printed code and have them explain each line:
  - Objective 2, Objective 3, Objective 4
- Give extra practice with lists
- Give extra practice with if:elif:else statements
- Give extra practice with logical operators
- Use the Code Tracing Chart for an objective
- Submit final “boundary.py” program

Success Criteria

- □ Use a threshold to determine if the ‘bot has reached a black line.
- □ Have the ‘bot respond to a black line by moving in a different direction.
- □ Use a list to store Boolean values for each sensor reading.
- □ Use default parameters in a function.
- □ Use multiple branching and compound conditions to have the ‘bot make smart decisions.

Vocabulary

- **Threshold:** A value halfway between two distinct sensor readings, like a white surface and black line.
- **Default parameter:** A parameter that uses a default value if an argument is not passed for it specifically.
- **Globals:** Variables defined outside a function. They are available throughout the entire program.
- **Parameter:** A named variable that is listed in a function definition; the variables receive values from arguments.
- **Argument:** Values passed when you call a function; the values correspond to parameters.
- **Positional argument:** An argument that must be passed to a parameter in the correct order; it is assigned to a parameter based on its position.
- **Keyword argument:** An argument assigned to a parameter by name instead of position.
- **Logical operators:** Operators that are used to compare multiple conditions. ‘And’ and ‘or’ are logical operators, and ‘not’ is a special kind of logical operator. When ‘or’ is used, at least one condition must be true for the compound condition to be true. When ‘and’ is used, all conditions must be true for the compound condition to be true.
- **Docstring:** A way to document functions using triple quotes at the beginning and end of the documentation comments.

New Python Code

`threshold = 2000`	Define a threshold.

`if val > threshold:`
`    break`
	Use threshold in an if statement.

`def go(left, right, delay=0)`
`def back_turn(turn_power=50)`
	Function definition with default parameter (delay and turn_power)

`if delay:`
`    sleep(delay`
	Use a default parameter in an if statement (acts like a Boolean)

`'''A function to simplify commands'''`	Docstring (commenting a function using triple quotes)
`sensors = [ ]`	Define an empty list
`sensors.append(is_line)`	Append the item in parenthesis to a list (sensors)
`leds.ls(vals)`	Use a Boolean list to control LEDs. In this example, vals is a list of Booleans. Example: vals = [True, True, False, False, False]
`if any(vals):`	Use the any() function in an if statement. Returns True if any item in the list is True.

`if vals[0] and not vals[4]:`
`    back_turn(30)`
`elif vals[4] and not vals[0]:`
`    back_turn(-30)`
`else:`
`    back_turn()`
	If:elif:else statement for multiple branches.
	This example also uses compound conditions with logical operators, and a function call with a default parameter.

Real World Applications

Many real-world applications have lines to mark boundaries. People stay in bounds when driving on roads, playing sports, and at places like school, warehouses and airports.

- Select a situation and describe an algorithm to help a person stay inside boundaries.
- What are some physical devices that must stay in boundaries? How do they accomplish this?

Teacher Notes

- Watch the indenting when calling the brake() function in Obj. 3. It is NOT indented in the while loop.
- The Debugger can be used to track variables. You can also use the Code Tracing Chart and have the class go through some code together, particularly in the later objectives.
- 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.
- Some new programming concepts are used in the later objectives. Students may need extra help and/or practice with them:
    - Appending items to an empty list.
    - Using a list of Booleans to control LEDs.
    - Using logical operators to form compound conditions.
    - Using an if:elif:else statement for multiple branching.
    - Default parameters.

Extensions / Cross-Curricular

- Add sound to the moving ‘bot.
- Add user LEDs to indicate what the ‘bot is doing, like moving forward, braking or turning.
- Add an animation when a boundary is reached before turning and moving or instead of turning and moving.
- **LANGUAGE ARTS:** Have students write a short poem about their sumo ‘bot.
- **SCIENCE:** In this Mission the ‘bot must brake and then turn. Discuss the physics involved.
- **SCIENCE:** Turn some of the objectives into experiments. Make an hypothesis and go through the scientific method to prove or disprove.
- **MATH:** Use different speeds for driving forward and different times for braking. Change just one value at a time and make a chart of the data. Then graph the data.

Mission 9: Line Following

Time Frame: 3-5 hours

Project Goal: Students will use sensor inputs to program the 'bot to follow a line course.

Learning Targets

- I can use the Debugger to track global and local variables.
- I can test code using REPL.
- I can define and use a dictionary.
- I can avoid KeyErrors by using get().
- I can define a function for a PID controller that continuously calculates the error value.
- I can define a function to collect stats on the error values.
- I can write code that makes the 'bot follow a complex path in under five minutes.

Key Concepts

- The REPL is a way to experiment with code.
- A tuple is a read-only list and is defined with parentheses instead of square brackets.
- A dictionary allows you to look up a value using a key, like a label.
- A default value can be returned if a key is not in the dictionary to avoid KeyError.
- A control loop mechanism that employs feedback by continuously calculating the error value is called a PID controller.

Assessment Opportunities

- Quiz after Objective 2
- Quiz after Objective 4
- Quiz after Objective 6
- Quiz after Objective 8
- Objective 5 Line Sensor Chart
- Use the Code Tracing Chart for an objective
- Give students printed code from an objective and have them explain each line
- Give extra practice with dictionaries
- Compare and contrast global / local variables
- Submit the "line_scanner.py" program
- Submit the "bang_bang.py" program
- Submit final "line_follower.py" program

Success Criteria

☐ Use previous knowledge of the motors and line sensing to make CodeBot follow a path.

☐ Use the Debugger and REPL to track variables and test code.

☐ Define and use a dictionary without errors.

☐ Define functions that create a PID controller.

☐ CodeBot follows a complex path:

- ☐ Does not veer off course
- ☐ Stays on the white board
- ☐ Finishes the course within time limit

Vocabulary

- **Global variables:** Variables defined outside a function. They are available throughout the entire program. If a global variable is changed inside a function, it must be declared in the function with "global".
- **Local variables:** Variables created inside functions; they only exist while the function is running and can only be used inside the function.
- **REPL:** Read Evaluate Print Loop – the command line that lets you type Python statements directly and observe what happens.
- **List comprehension:** A shortcut for appending items to a list using a for loop inside the square brackets.
- **Tuple:** An immutable sequence of items that you access with an index. It is similar to a list, but the items in the tuple cannot be changed or added to; it is read-only. Use parentheses to define a tuple instead of square brackets.
- **None:** 'None' means no value, or null. It is a special object with type 'nonetype'.
- **Dictionary:** A container that holds keys and values. It supports fast lookup of a value based on a given key. Dictionaries are defined with curly braces { }
- **KeyError:** Result when a key is given that is not in a dictionary.
- **get() method:** Result when a key is given that is not in a dictionary.
- **PID controller:** A control loop mechanism employing feedback that is widely used in industrial control systems by continuously calculating the error value.

New Python Code

```
even_numbers = [x*2 for x in range(5)]
sensors=[ls.read(i)>2000 for i in
        range(5)]
```

List comprehension
(a shortcut for populating a list)

`vals = ls.check(2000)`	Function that reads all line sensors and compares the reading to a threshold, then returns a Boolean list
`if vals != prev_vals:`	‘Not equals’ operator is !=
`elif vals[1] or vals[2] or vals[3]:`	Using a logical operator in a compound condition

`ls_err = {`
`    (0,0,1,0,0) : 0,`
`    (0,1,1,1,0) : 0,`
`}`
	Define a dictionary with a key and value.
	The syntax is key : value
	This example uses a tuple as the key and an integer as the value.

`err = ls_err[vals]`	Look up a value from a dictionary using a key (vals)
`err = ls_err.get(vals, err)`	Using the get() method to avoid KeyError

`def apply_conrol(err):`
`    Kp = 0.1`
`    steering = err * Kp`
`    drive(SPEED, steering)`
	PID controller
	Initial function, before adding in stats.

Real World Applications

Staying on the path is the staple of self-driving cars, autonomous flying drones and other computing systems that navigate on their own.

- Line followers are a staple of robotics competitions, like races or mazes.
- Robots zip through warehouse distribution centers following lines.

The coding skills learned in this Mission can be used in industrial and commercial robotics applications. They are also useful for school and club robotics competitions.

Teacher Notes

- When using the debugger, expand the list for Globals and Locals by clicking on the > to see them. You can expand the window panel.
- You can also use the Code Tracing Chart and have the class go through some code together, Obj 1 and/or Obj 4 are good choices.
- Objective 3 still works on Obj 4 without making the changes. But this is a good place for experimenting. Have students try making the speed faster when a line is detected (last elif branch) and see what happens.
- A document for Objective 5 Line Sensor Readings is available at learn.firialabs.com.
- For the final objective, students need to tweak the speed and/or other constants. Encourage students to change one variable at a time and keep track of the changes so they can return to the previous value if needed. Small changes can make a big difference.
- Some new programming concepts are used in the later objectives. Students may need extra help and/or practice with them:
    - Dictionaries
    - Global and local variables

Extensions / Cross-Curricular

- Have a competition to see who’s ‘bot can reach the finish line the quickest.
- Add sound to the moving ‘bot. Vary the pitch according to speed or turning angle.
- Set user LEDs to indicate the speed of the ‘bot.
- Set up a track and try the code on a physical CodeBot.
- **LANGUAGE ARTS:** Have students compare and contrast how the line sensors are used in Mission 8 and Mission 9. What are the similarities and differences?
- **SCIENCE:** Have a lesson on PID controllers.
- **MATH:** The equation for PID uses an integral and derivative. If your students are in calculus, go over the math.
- **MATH:** The mission uses statistics to inform the speed and turning ratio of the ‘bot. Go over the math involved.
- **MATH:** The PID controller uses proportions and ratio to affect the speed of each wheel. Discuss how it relates to the code.

Unit 3 Remix Project and Exam	**Time Frame:** 2-5 hours
Project Goal: Students will use the skills and concepts they learned in Mission 7, 8 and 9 to create their own project. **Project Outline:** Follow the five-steps of the design process (document on next page) to design a remix project.	**Exam Goal:** Students will show mastery of Python and robotics concepts by obtaining at least 75% correct on a multiple choice test. **Project Outline:** Review concepts through additional practice and Kahoots. Then students take the exam. An option for Microsoft Forms is available. Vocabulary and concepts are two different tests.
Remix Project Assessment Opportunities ● Unit 3 Remix Project Planning Guide ● Peer reviews / Gallery walk ● Unit 3 Remix Project Rubric Checklist ● Submit Unit 3 Remix Project Program	**Unit 3 Exam Opportunities** ● 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)

Supplementary Materials (available at learn.firialabs.com)

- Unit 3 Remix Project Planning Guide
- Unit 3 review and test questions
- Code Tracing Chart

Unit 3 Remix Project Ideas:

- Pick an extension idea from Mission 7, 8 or 9.
- Create a program similar to the line sensor program (Mission 7) but use a dictionary instead of a matrix. Add extra compass points where the left and right sensor have the same value.
- Add button input. Use button 0 to start the motors and button 1 to stop the motors.
- Combine two missions into one program, and use a button press for input – when button 0 is pressed, one mission is accomplished, and when button 1 is pressed the other mission is accomplished.
- Combine all three missions into one program. Get user input from the console to select the mission to accomplish.
- Think of your own creative project with line or proximity sensors, movement, LEDs and button input.

Unit 3 Remix Project Rubric Checklist:

☐ Filename is descriptive
☐ Uses global and local variables appropriately
☐ Uses one or more constants, each with a descriptive name
☐ Uses line sensor data to control CodeBot's movement
☐ Uses LEDs to communicate information (user LEDs and/or line sensor LEDs)
☐ Defines and calls at least one function
☐ Uses a list or dictionary
☐ Gets input from the user (button press, input() function)
☐ 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: Applications (6-12 hours)

Students learn a lot more about a Python dictionary and how to use it in an application. They also apply the line sensor readings in an application that involves math operations.

Summary of Mission 10:

In this mission students will control a robot dog by compiling a dictionary of commands and writing functions for each action. Students then use the commands to move the 'bot around the floor collecting treats and getting a high-five.

Summary of Mission 11:

The CodeBot uses a line follower program to perform tasks on an airfield runway. Students will learn additional math operations that can be used for automated tasks in this useful application.

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)

Standards addressed in this unit:

CSTA Standards Grades 9-10	CSTA Standards Grades 11-12
● 3A-CS-01 ● 3A-CS-02 ● 3A-CS-03 ● 3A-DA-09 ● 3A-DA-10 ● 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-22 ● 3A-AP-23 ● 3A-AP-26	● 3B-CS-02 ● 3B-DA-05 ● 3B-DA-06 ● 3B-DA-07 ● 3B-AP-10 ● 3B-AP-11 ● 3B-AP-12 ● 3B-AP-14 ● 3B-AP-15 ● 3B-AP-16 ● 3B-AP-17 ● 3B-AP-20 ● 3B-AP-21 ● 3B-AP-22 ● 3B-AP-23

Mission 10: Fido Fetch

Time Frame: 2-3 hours

Project Goal: Students will train the CodeBot to follow commands using a Python dictionary and console inputs.

Learning Targets

- I can give input through the console.
- I can define and add key:value command pairs to a dictionary.
- I can use function calls as values in a dictionary.
- I can iterate over a dictionary.
- I can use a dictionary of commands to control a robot dog.

Key Concepts

- A dictionary has several benefits. You can search by key or value, and looking up values in a dictionary is more efficient than searching through a list of items to find a match.
- You can add key:value pairs to a dictionary after it has been defined.
- You can delete a key:pair from a dictionary.
- You can iterate, or traverse, over a dictionary.

Assessment Opportunities

- Quiz after Objective 3
- Quiz after Objective 7
- Use the Code Tracing Chart for an objective
- Give students printed code from an objective and have them explain each line
- Give extra practice with dictionaries
- Submit final “fido.py” program

Success Criteria

□ Use the input() function to give a command.

□ Define a dictionary with different key:value pairs.

□ Add key:value command pairs to a dictionary.

□ Use functions as values in a dictionary.

□ Write an efficient program without using an if:elif statement for looking up values.

□ Use dictionary commands to control the robot dog (collect treats, get a high-five).

Vocabulary

- **Dictionary:** A container that holds keys and values. It supports fast lookup of a value based on a given key. Dictionaries are defined with curly braces { }
- **KeyError:** Result when a key is given that is not in a dictionary.
- **Refactor:** Improving the structure of your code by making major changes to the code while accomplishing the same task.

New Python Code

`value = dictionary[key]`
`response = commands[command]`
	Look up the value of a key:value pair

`dictionary[‘key’] = value`
`commands[‘come’] = [30, 30]`
	Add a key:value pair to a dictionary that is already defined

`for k in commands:`
`    print(k)`
	Iterate over a dictionary (k stands for ‘key’).
	This example prints all keys to the console.

`del_key = input(‘Command to forget: ‘)`
`del commands[del_key]`
	Remove a key:value pair from the dictionary.

Real World Applications

Dictionaries are used in all types of software. They are an efficient way to search collections of items you define in code, or for looking up items for a simple task. Think about objects that complete a task after you tell it what to do. Things like coffee makers and microwave ovens. Each has several tasks to accomplish, and you input which one you want it to do.

- What other items can you think of?
- How can they use a dictionary to be efficient?

Teacher Notes	Extensions / Cross-Curricular
● This Mission is pretty straightforward. It is all about dictionaries. Review concepts or give extra practice as needed. ● During this mission is a good time to review and re-emphasize abstraction in code. With the addition of dictionaries, students have a variety of tools for abstraction: functions, lists and dictionaries. Compare levels of abstraction in hardware and software. ● Remember that you must zoom into the environment to hear the speaker. ● If needed, review accessing items from a list using index for Objective 4. ● Meeting the last objective will be different for each student. When a “treat” is taken by the ‘bot, a small green check will appear over it. Some trial and error and practice will be needed, but there is no time limit. If a student gets frustrated, just have them start over. ● For the high five goal, the ‘bot just needs to move very fast. The ‘bot can do it first, or last, or any time.	● Add more commands and functions. You could incorporate LEDs, or have different barks, and even have actions like ‘rollover’. ● Use a button to wake up the dog and have it ready for commands. ● Use a button to quit the program – the dog needs to take a nap! ● **LANGUAGE ARTS:** Have students write a short story about a robot dog. ● **MATH:** The final objective requires the robot dog to move around the floor and collect treats. A lot of angles and distances are involved. Use math to determine the path for the robot dog to take.

Mission 11: Airline Ops

Time Frame: 2-4 hours

Project Goal: Students will learn math operations that help with airfield runway operations.

Learning Targets

- I can use line sensor data to detect a white line.
- I can use line sensor data to count dashed lines.
- I can use the number of dashed lines with math operations //, % and **.
- I can gather data and use the data to accomplish tasks.

Key Concepts

- CodeBot can gather data, like line sensor readings, and use it to accomplish tasks.
- Besides addition, subtraction, multiplication and division, other math operators are available and useful in programming:
    - // integer division
    - % modulo (remainder)
    - ** power (exponent)

Assessment Opportunities

- Quiz after Objective 6
- Use the Code Tracing Chart for an objective
- Use the Debugger to track variables for each math operation (Obj 4, 5, and 6)
- Give students code snippets from an objective and have them explain what it does
- Give extra practice with the math operations
- Submit final runway clearing program

Success Criteria

- ☐ CodeBot follows a dotted white line.
- ☐ CodeBot stops at the end of the white line.
- ☐ CodeBot uses LEDs as a position indicator.
- ☐ CodeBot makes a scary sound every 8 dashes.
- ☐ CodeBot's proximity LEDs light up for each red marker.

Vocabulary

- **Increment :** Adding 1 to a variable (like counting)
- **Decrement:** Subtracting 1 from a variable (like counting down)
- **Augmented assignment:** A shorter way to write common expressions
- **// operator:** Integer division, or quotient; divides a number and then truncates the answer to an integer
- **% operator:** Modulo – gives the integer remainder of a division problem
- ** **operator:** Gives the power of the base number by the exponent

New Python Code

`count = count + 1`	Increment a counter
`count += 1`	Augmented assignment for incrementing

`num = dividend // divisor`
`leds=(count*LEDS)//TOTAL`
	Integer division
	(returns the quotient, or truncated answer as an integer)

`progress=[True]*leds_on`	Use multiplication on a Boolean list with an integer
`remainder = count % 8`	Module; returns the remainder of a division problem. In this example, all possible answers are 0, 1, 2, 3, 4, 5, 5, or 7.
`marker_dash = 2**next_marker`	Power (or exponent) operator. In this example, 2 will be raised to the next_marker power.

Real World Applications

Modulo and integer division are very useful mathematical operations. They can be used in a variety of applications and automations. In this program, math integer division and modulo were used to turn on LEDs and detect every 8th line. It can be used in packaging, keeping track of player turns, and much more.

- What applications can you think of that use integer division and/or modulo?
- What devices and tasks could benefit from integer division and/or modulo?

Teacher Notes

- This mission is another line follower program, with applications. The CodeTrek shows code for a new program. You can make a choice. If you want students to practice the line follower algorithm, and have it more simple than Mission 9, they can follow CodeTrek and start new code. However, they can also use their former code (either Mission 8 or Mission 9) and add to it to meet the goals for Mission 10.
- If students use a previous program for this Objective, they just need to change the line sensor condition from a black line to a white line.
  > `ls.check(threshold, is_reflective)`
  > For white line: **ls.check(threshold, True)**
- For the other objectives, the code modifications and functions can be easily added to a previous program, or to a new program.
- The value of TOTAL_LINES may need to be adjusted to meet the goal.

Extension / Cross-Curricular

- Use button input. Button 0 to start the automation, and button 1 as an emergency stop button.
- Use a dictionary of commands, like start, stop or scary sound. Use console input to control the ‘bot.
- Think of additional tasks for the ‘bot to perform on the airfield.
- **LANGUAGE ARTS:** Have students write a description of integer division and modulo. Or have them create a worksheet with instructions that other students could use.
- **MATH:** The mission is all about math! Students learn about integer division and modulo. Have a lesson about this type of division, and discuss when you might want to use it instead of regular division. Have students complete practice problems.

Unit 4 Remix Project and Exam

Time Frame: 2-5 hours

Project Goal: Students will use the skills and concepts they learned in Mission 10 and Mission 11 to create their own project.

Project Outline: Follow the five-steps of the design process (document on next page) to design a remix project.

Exam Goal: Students will show mastery of Python and robotics concepts by obtaining at least 75% correct on a multiple choice test.

Project Outline: Review concepts through additional practice and Kahoots. Then students take the exam. An option for Microsoft Forms is available. Vocabulary and concepts are two different tests.

Remix Project Assessment Opportunities

- Unit 4 Remix Project Planning Guide
- Peer reviews / Gallery walk
- Unit 4 Remix Project Rubric Checklist
- Submit Unit 4 Remix Project Program

Unit 3 Exam Opportunities

- 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)

Supplementary Materials (available at learn.firialabs.com)

- Unit 4 Remix Project Planning Guide
- Unit 4 review and test questions
- Code Tracing Chart

Unit 3 Remix Project Ideas:

- Pick an extension idea from Mission 10 or Mission 11.
- Modify the airfield ops program to use a dictionary of functions. Look up the function to call during the automation.
- Combine the two missions into one program, and use a button press for input – when button 0 is pressed, use fido fetch, and when button 1 is pressed, complete the airfield ops. Organize each mission into functions.
- Combine parts of each mission into one program. Use a dictionary of commands, and one of the commands can be to complete the airfield ops. Add more commands and functions. Use math operations for at least one more command. Count how many times a certain command was requested, and change the result based on the count or modulo.
- Think of your own creative project with line or proximity sensors, movement, LEDs and button input.

Unit 4 Remix Project Rubric Checklist:

- □ Filename is descriptive
- □ Uses global and local variables appropriately
- □ Uses at least one concept from Mission 10:
    - □ Define and use a dictionary
    - □ Iterate over a dictionary
    - □ Add a key:value pair to a dictionary
- □ Uses at least one concept from Mission 11:
    - □ Increment a counter
    - □ Math operations: //, %, **
    - □ Control LEDs with a Boolean list
- □ Defines and calls at least one function
- □ Gets input from the user (button press, input() function)
- □ Includes something extra (sound, more than one sensor, more than one function, more LEDs, etc.)
- □ Code follows programming conventions of comments, readability, indenting, and capitalization
- □ Code runs with no errors

Unit 5: Underneath the Hood (7-13 hours)

Students become familiar with additional sensors on the CodeBot: the accelerometer and wheel encoders. They learn to read the accelerometer's data and convert it into usable information, like radians or degrees. They use wheel encoders to go exact distances and speeds.

Summary of Mission 12:

In this mission students learn to read the accelerometer's data and convert it into usable information, like radians or degrees. Then they use the data to control the 'bot remotely. They refactor the code so the CodeBot drives autonomously up and down a hill with obstacles. Students also print data using string formatting techniques.

Summary of Mission 13:

The CodeBot's wheel encoders allow you to directly program a given distance to travel or speed to maintain, without regard to surface or battery power. Students learn about the wheel encoders and how they work. Then they use wheel encoders to calculate the number of slots (or counts) needed to go a specific distance. With a little bit more math, the same data can be used to maintain a constant speed, either in a straight line or moving in an arc.

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)

Standards addressed in this unit:

CSTA Standards Grades 9-10	CSTA Standards Grades 11-12
● 3A-CS-01 ● 3A-CS-02 ● 3A-CS-03 ● 3A-DA-10 ● 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-22 ● 3A-AP-23 ● 3A-AP-26	● 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-20 ● 3B-AP-21 ● 3B-AP-22 ● 3B-AP-23

Mission 12: King of the Hill

Time Frame: 2-3 hours

Project Goal: Students will use data from the accelerometer to “Master the Hill.”

Learning Targets

- I can print accelerometer data to the console in three different ways.
- I can use internal sensors to monitor physical orientation by converting data to angles.
- I can create a data visualization using string replacement fields, format specifiers and escape sequences.
- I can control the ‘bot using buttons for input.
- I can program the ‘bot to make decisions based on data.

Key Concepts

- The accelerometer detects orientation in three dimensions.
- Data can be printed as a string using formatting such as replacement fields indicated with { } and format specifiers.
- Special characters can be displayed using escape sequences.
- Data from the accelerometer can be converted to angles, either in radians or degrees. This data can then be used to make decisions.
- The CodeBot can be programmed to act on conditions based on its orientation.

Assessment Opportunities

- Quiz after Objective 1
- Quiz after Objective 5
- Use the Code Tracing Chart for an objective
- Use the Debugger to track variables
- Give students code snippets from an objective and have them explain what it does
- Give extra practice with string formatting
- Submit CodeBot remote control program (Objective 4 or Objective 6)
- Submit CodeBot autonomous program (Obj. 8)

Success Criteria

□ Read and print data from the accelerometer.

□ Convert accelerometer data to an angle using trigonometry.

□ Use button presses as input to remote-control the CodeBot.

□ Display a bar graph visualization of the data.

□ Program CodeBot to drive autonomously up and down a hill with obstacles.

Vocabulary

- **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
- **Vector:** A mathematical or geometric representation of magnitude and direction. Examples of vectors in nature are velocity, momentum, force, and weight.
- **Math module:** A scientific calculator for your code; a module that includes a set of mathematical functions and constants.
- **Replacement fields:** A template for formatting a string by using { } to designate where and how to print the values of variables.
- **Format specifiers:** Information added inside the replacement field that dictates how to display a number.
- **Escape sequences:** A way to insert special characters in a string; begins with a backslash \
- **Hexadecimal:** Base 16. A single hex digit holds exactly 4-bits of information and ranges from 0 to 15. In an escape sequence, a hex value is designated with \x
- **Autonomous:** A device that can sense its environment and respond to change.

New Python Code

`X, y, z = accel.read()`	Read the current axis values of the accelerometer
`accel.dump_axes()`	Print the 3-axis values to the debug console The display: X=-1044, Y=4261, Z=-15786
`pitch = math.asin(y/16384)`	Calculate the radians from accelerometer data
`pitch = pitch * 180 / math.pi`	Convert radians to degrees

`pitch = round(pitch)`	Round a value to the nearest integer
`bars_left = bars_right = ‘’`	Cascaded assignment; assigns two variables to the same value
`dash=”[-90 {:>30}]”.format(bars_left)`	Formatted string with replacement field and format specifier
`+90\xB0`	Use escape sequence and hex to display degrees

Real World Applications

Accelerometer data can be converted into angles. It can be used to compute the pitch, roll and yaw. This data is used to navigate ships, aircraft and robots. It is a key element of autonomous devices.

- Discuss devices that have an accelerometer and how it is used to make decisions.

This mission showed how to visualize data with a type of bar chart. Data visualization is used in many devices.

- Discuss data visualization in a variety of devices, from cars, to video game displays, battery indicators, etc.

Teacher Notes

- Students will drive the ‘bot using the 0 and 1 number keys instead of pressing the ‘bot’s buttons. They can press both buttons at the same time to move forward, or press a single button to turn.
- For Objective 8, just a slightly larger turn value for one wheel makes a big difference for reaching the goals.
- This mission uses many string formatting techniques. This is a good time to do some practice and review.
- Each objective builds on the last, adding code or making modifications. You might want to pause after every couple of objectives and review the new concepts before moving forward.

Extension / Cross-Curricular

- Use the remote-control program on a physical CodeBot and a sloping surface.
- Use the autonomous program on a physical CodeBot and obstacle course.
- Add LEDs and sounds to indicate CodeBot progress. You could count every obstacle encountered, or keep track of time or distance and indicate forward progress. Add beeps for obstacle detection or when backing up.
- **LANGUAGE ARTS:** Have students write a summary of their mission. Explain the purpose and function of the program, and who the intended users are.
- **MATH:** The mission uses trigonometry and angles. Discuss radians and degrees. Have a lesson on the Pythagorean theorem. There are many math applications with this lesson.
- **SCIENCE:** The accelerometer measures gravity. Have a lesson on gravity, and maybe in space and on other planets.

Mission 13: Going the Distance

Time Frame: 3-5 hours

Project Goal: Students will use wheel encoders to make the CodeBot move exact distances and angles.

Learning Targets

- 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.
- I can write a function that calculates the speed of the 'bot from wheel encoder data.
- I can write a function that enables the 'bot to maintain a speed.
- I can write a function that reads both wheel encoders and uses the data to move in an arc at a given speed.

Key Concepts

- 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.
- The data from the wheel encoders can be used to maintain a given speed.

Assessment Opportunities

- Quiz after Objective 3
- Quiz after Objective 6
- Quiz after Objective 10
- Use the Code Tracing Chart for an objective
- Use the Debugger to track variables and/or lists
- Give students code snippets from an objective and have them explain what it does
- Give extra practice with math calculations for distance
- Submit the "encoder_check.py" program
- Submit the "enc_drive.py" program
- Submit final "enc_speed.py" program

Success Criteria

- ☐ Write code to create a bar chart visualization for wheel encoder data.
- ☐ Define a drive() function to move CodeBot an exact distance, based on the counts of the wheel encoder.
- ☐ Define a rotate function that turns the 'bot a specified angle, based on wheel encoder data.
- ☐ Calculate the speed of CodeBot and display the data in a bar chart visualization.
- ☐ Write "cruise control" code to maintain a set speed over any terrain.
- ☐ Define an arc function to drive the 'bot around a free throw ring.

Vocabulary

- **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.
- **Integer division:** The // operator, which gives the quotient
- **Wheel track:** The circular path wheels take when the 'bot rotates in place:
  Distance = circumference (angle / 360)
  For clockwise, use a positive left and negative right wheel power.
  For counterclockwise, use a negative left and positive right wheel power.
- **Speed:** Distance / Time (can by any measures; we will use cm/second and also counts/second)
- **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.

New Python Code

`val = enc.read(LEFT)`	Reads the encoder's analog value (can be LEFT or RIGHT)
`if enc_state != slot:`	The != is the "not equals" comparison operator

`n = val // 100`	Integer division (// operator)
`print(val, n * “*”)`	Create a repeated character string. In this example, “n” number of “*” is printed.

`val = enc.read(LEFT)`
`is_slot = val > SLOT_THRESH`
	Use a Boolean to indicate a slot (True or False)

`WHEEL_CIRC_CM = math.pi * WHEEL_DIAM_CM`	Calculate the wheel circumference
`counts += 1`	Augmented assignment to update (increment) counts
`cm * (COUNTS_PER_REV / WHEEL_CIRC_CM)`	Calculate the counts needed for a given distance
`direction = deg / abs(deg)`	Get a positive or negative direction (-1 or +1)
`t_poll = ticks_ms() + POLL_MS`	Current time-tick count in milliseconds (ticks_ms starts at 0)
`print(f’speed = {speed:.1f} cm/s’)`	F-string formatting with one decimal place for float
`counts * WHEEL_CIRC_CM / COUNTS_PER_REV`	Calculate distance for the given counts

`err = target_speed - cur_speed`
`power += err * Kp`
`motors.run(LEFT, power)`
`motors.run(RIGHT, power`
	Feedback loop

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.

Cruise control! During this mission you learned to have the ‘bot automatically maintain a speed. The same sort of feedback loops are used to control all kinds of motors, heaters, and other systems you use daily in the *real world*.

Teacher Notes

- For Objective 8, the number given in CodeTrek for the distance is a starting point. It will need to be changed to meet the goal.
- For Objective 11, remember to change the arguments for drive_speed(). The values given in CodeTrek will not meet the goal.
- For Objective 12, the global designation is only needed in the init_drive_state() function. Lists are global and can be updated anywhere. The function needs the global designation because the lists originate there and cannot be local.
- You may need to adjust values to meet the goal. Slower speeds work better.
- There is a lot of information given in this lesson. 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.

Extensions / Cross-Curricular

- Use the drive or the speed program on a physical CodeBot to run a track or maze.
- Add LEDs and sounds to indicate CodeBot progress. Use LEDs or beeps to indicate speed or a turn.
- **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.

Unit 5 Remix Project and Exam

Time Frame: 2-5 hours

Project Goal: Students will use the skills and concepts they learned in Mission 12 and Mission 13 to create their own project.

Project Outline: Follow the five-steps of the design process (document on next page) to design a remix project.

- **Exam Goal:** Students will show mastery of Python and robotics concepts by obtaining at least 75% correct on a multiple choice test.
-
- **Project Outline:** Review concepts through additional practice and Kahoots. Then students take the exam. An option for Microsoft Forms is available. Vocabulary and concepts are two different tests.

Remix Project Assessment Opportunities
- Unit 5 Remix Project Planning Guide
- Peer reviews / Gallery walk
- Unit 5 Remix Project Rubric Checklist
- Submit Unit 5 Remix Project Program

Unit 5 Exam Opportunities
- Unit 5 Vocabulary review Kahoot
- Unit 5 Concepts and coding review Kahoot
- Unit 5 Vocabulary test (MS Form)
- Unit 5 Concepts & coding test (MS Form)

Supplementary Materials (available at learn.firialabs.com)
- Unit 5 Remix Project Planning Guide
- Unit 5 review and test questions
- Code Tracing Chart

Remix Ideas:
- Pick an extension idea from Mission 12 or Mission 13.
- Combine the accelerometer readings with the navigation system for specific driving capabilities over different surfaces, like the “hill” in Mission 12.
- Use buttons as input. They can be used to start and stop the ‘bot, or to switch from the ‘bot going a specific distance to going a specified speed.
- Add a function for turning counterclockwise. Use buttons to indicate which way to turn.
- Use the accelerometer to determine turning angle instead of the wheel encoders and have the ‘bot travel a path or maze.
- Think of your own creative project with line or proximity sensors, movement, LEDs and button input.

Rubric Checklist:

☐ Filename is descriptive
☐ Uses global and local variables appropriately
☐ Reads one or more sensors: wheel encoder 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
☐ Uses at least one list or tuple for data from a sensor
☐ CodeBot drives to a given destination
☐ Includes something extra (sound, more than one sensor, more than one function, both buttons, etc.)
☐ Code follows programming conventions of comments, readability, indenting, and capitalization
☐ Code runs with no errors

Unit 6: Synthesize (6-11 hours)

The missions in this unit synthesize student learning by using many previous programming techniques and concepts. The main focus of the unit is using files. Students learn a variety of file operations and techniques for manipulating the data from files to accomplish a task. The missions in the unit show two different applications for Python programming.

Summary of Mission 14:

Students learn about file operations while creating a CodeBot jukebox. The mission uses dictionaries and lists to store information about notes and songs. This information can be stored in a file. For loops are used to extract information and play songs.

Summary of Mission 15:

Students create an email detection system by using file operations and string methods. Previous coding concepts, like dictionaries, lists and for loops, are used with files. The CodeBot is not used in this Mission; instead data is printed to the Console.

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)

Standards addressed in this unit:

CSTA Standards Grades 9-10	CSTA Standards Grades 11-12
● 3A-CS-01 ● 3A-CS-02 ● 3A-CS-03 ● 3A-NI-07 ● 3A-NI-08 ● 3A-DA-10 ● 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	● 3B-CS-02 ● 3B-NI-04 ● 3B-DA-05 ● 3B-AP-10 ● 3B-AP-12 ● 3B-AP-14 ● 3B-AP-15 ● 3B-AP-16 ● 3B-AP-17 ● 3B-AP-18 ● 3B-AP-20 ● 3B-AP-21 ● 3B-AP-22 ● 3B-AP-23

Mission 14: Music Box

Time Frame: 2-3 hours

Project Goal: Students will turn the CodeBot into a jukebox and learn about Python’s file operations.

Learning Targets

- I can use a dictionary of notes to play different pitches.
- I can use a list of notes to play a song by accessing the dictionary of notes.
- I can open, read, write to, and close a text file.
- I can play a tune using a matrix of notes and beats.
- I can convert string data from a file into a matrix.

Key Concepts

- A dictionary of notes is useful for storing data that is used frequently.
- The Python file system allows you to open, read and write to a file in many ways.
- You should always close or flush a file after it is opened.
- String data can be converted to an integer.
- String data from a file can be converted to a list of lists using file operations.

Assessment Opportunities

- Submit program code, or give students printed code and have them explain each line:
  - Objective 3, Objective 6, Objective 8
- Give extra practice with file operations
- Give extra practice with iterating over a list
- Use the Code Tracing Chart for an objective
- Submit final jukebox program

Success Criteria

- ☐ Create a dictionary of notes and pitches.
- ☐ Iterate over a list of notes and beats to play a song.
- ☐ Open, read and close a file.
- ☐ Convert string data from a file into a matrix (list of lists).
- ☐ Program CodeBot to cycle through songs like a jukebox when a button is pressed.

Vocabulary

- **Scientific Pitch Notation:** Writing musical notes in the form C$_5$ or C5. The letter is the note, and the number is the octave. C$_4$ is middle C on the piano.
- **split() function:** Turn a string into a list.
- **File operations:** Reading and writing, and using files in a Python program.
- **Mode:** The way a file is accessed, such as read-only, read and write, write-only, etc.
- **Flush:** Guarantee that any buffered data is saved to the filesystem (close or flush function).
- **Matrix:** Multidimensional list (lists inside a list).
- **int() function:** Takes a value (string or float) and converts it to an integer.

New Python Code

`notes = [‘C’, ‘G’, ‘A’, ‘E’, ‘F’]`
`for note in notes:`
`    f = freqs[note]`
`    spkr.pitch(f)`
`    sleep(beat_duration)`
	Iterate over a list of strings

`text = ‘E E G C D E F A’`
`notes = text.split()`
	Split a string into a list

`f = open(‘song_name’, ‘r’)`	Open a file – read-only
`text = f.read()`	Read the data from a file
`f.close()`	Close a file
`f = open(‘new_song’, ‘w’)`	Open a new file to write to

`f.flush()`	Flush a file (all data is saved before closing)

`song = [`
`    [‘E’, 1],`
`    [‘D’, 1],`
`    [‘C’, 2]`
`]`
	Create a matrix of notes and beats
	(list of lists)

`for note, beats in song:`
`    f = freqs[note]`
`    spkr.pitch(f)`
`    sleep(beats * beats_duration)`
	Unpack a matrix using a for loop

`beats = int(str_beats)`	Convert a string to an integer

`f = open(‘another_song.csv’, ‘r’)`
`file_lines = f.readlines()`
	Get a list of strings one from each line in a file

`song = []`
`for line in file_lines:`
`    note_beat = line.split(‘,’)`
`    song.append(note_beat)`
	Split a list of strings into a list of lists (comma delineator)

Real World Applications

Programmers write code all the time that pulls other files they or someone else created. Data can come from many sources, and collecting the data into a file for sharing and applications is something done in the real world.

- Discuss sources of data.
- Discuss how and where files of data may be used.

Teacher Notes

- Remember to zoom in enough to hear the speaker.
- When typing a long list, it can be divided up into more than one line, just like a dictionary, for easy reading.
- Make sure students include the file extension in the name for the open command.
- Programming concepts used are: dictionaries, iterating over a list, using a matrix, and file operations. Review them as needed.

Extension / Cross-Curricular

- Use a button press to scroll forward, and the other button to scroll backward.
- Add a light show by using LEDs while a song is played. You can even have a different light show for each song.
- Add CodeBot dances to the songs.
- Use a button to speed up the beat duration and make songs play faster.
- **LANGUAGE ARTS:** Have students explain how lists and dictionaries help manage the complexity of the program and how their program would be different without them.
- **PERFORMING ARTS:** Have students share music from their culture or heritage.
- **MATH:** The mission pitch for each note. Use the chart from Objective 1. Graph the frequency for each note. Then extrapolate information for other notes.
- **SCIENCE:** Have a lesson on sound waves.

Mission 15: Cyber Storm

Time Frame: 2-3 hours

Project Goal: Students help protect an email server by using file operations.

Learning Targets

- I can open and read from a file.
- I can check for special characters.
- I can create a dictionary from an email file.
- I can search for a specific word and replace it with a different word.
- I can create a list of blocked users and remove their emails.

Key Concepts

- File operations include a variety of ways to open, read, write and append to a file.
- The with statement is a for loop for files that automatically closes the file.
- Use ‘in’ or ‘not in’ to search through a string.
- A string can be manipulated with methods such as ‘replace’.
- A new file can be created and data added to the file using append.

Assessment Opportunities

- Quiz after Objective 4
- Quiz after Objective 8
- Use the Code Tracing Chart for an objective
- Submit program code, or give students printed code and have them explain each line:
  - Objective 3, Objective 5, Objective 7, Objective 8
- Give extra practice with file operations
- Give extra practice with string operations
- Submit final blocklist program

Success Criteria

- □ Open and read data from a file.
- □ Create a dictionary of an email’s heading and body.
- □ Search the body of an email for suspicious keywords.
- □ Create a blocklist file and use it to remove suspicious emails.

Vocabulary

- **‘with’ statement:** A type of for loop for files that automatically closes the file.
- **Escape sequence:** A way to insert a special character using \. Common escape sequences are:
- ‘\n’ for new line, ‘\r’ for carriage return, ‘\t’ for tab
- **strip() function:** A function that removes characters from the beginning and end of a string.
- **Whitespace:** Whitespaces include a space, a tab, a carriage return and a new line (escape sequences).
- **Concatenation:** Appending (or joining) a string.
- **startswith(prefix):** The function returns True if the string starts with the given prefix.
- **‘in’ and ‘not in’:** Used in a for loop to check if a value exists (or doesn’t exist) in a sequence.
- **replace() function:** Takes two arguments and replaces the first argument with the second in a string.
- **Blocklist:** A list of disallowed senders.

New Python Code

<table>
<tr><td><pre>with open(email_file, ‘r’) as f:
   file_contents = f.readlines()
   print(file_contents)</pre></td><td>Open a file using a ‘with’ loop</td></tr>
<tr><td><pre>line.strip()</pre></td><td>Strips whitespace from a line</td></tr>
<tr><td><pre>email = email + line.strip()</pre></td><td>Concatenate (or join) a string</td></tr>
<tr><td><pre>with open(email_file, ‘r’) as f:
   email = {}
   for line in f:
      if line.startswith(‘Date: ‘):
         email[‘date’]=clean_line[6:]
      elif line.startswith(‘From: ‘):
         email[‘from’]=clean_line[6:]
 … and so forth …</pre></td><td>Create a dictionary with the email heading</td></tr>
</table>

`elif line.strip() == ‘’`
`email[‘body’] = f.read()`
	Find the body of the email
	(use at the end of the heading if:elif statement)

`if ‘virus’ in email[‘body’]:`	Look for a word in a string
`email[‘body’].replace(‘virus’, ‘REMOVED’)`	Replace a word in a string
`import os`	Import a library to use os functions and methods

`if not os.path.exists(‘blocklist.csv’):`
`    print(“Creating blocklist!”)`
	Check to see if a file exists

`f.write(bl_entry)`	Write a new item to a file

`if sender in blocklist:`
`    os.remove(filename)`
	Remove a file

Real World Applications

This mission also uses files and file operations. Programmers write code all the time that searches file and creates new files.

- Other than cybersecurity, discuss other applications for searching files and creating new files.

Protecting data is extremely important in today’s world. Searching for suspicious emails is one way to avoid malware.

- Discuss other ways to protect your computer and your data.

Teacher Notes

- This mission does not use the CodeBot. It uses the console for displaying information.
- Have students do a “save as” at the beginning of each objective, so they can refer back to methods taught earlier.
- The name of the file used in the program changes frequently from one objective to another. Make sure you are using the correct filename each time.
- This mission uses many file operations. This is a good time to do some practice and review.
- There are several CSTA standards that are specific to networks, privacy and security. Before or after this mission is a good time to have additional lessons and discussions on these topics.

Extension / Cross-Curricular

- Use CodeBot with the email protection program. Add LEDs or sounds to indicate a potential malicious email.
- Use a button press to start the email protection program.
- Use the console to get input from the user, like a filename or malicious word to search for.
- Add a list of malicious words to search for instead of a single word.
- **LANGUAGE ARTS:** Have students describe in detail how one of their functions works.
- **MATH:** The clean_line function can create a substring of a string using each letter’s position (or index). Create an algorithm or equation that automates this process without needing ‘magic numbers.’
- **SCIENCE:** The mission checks for a potential virus that could infect a computer. Discuss viruses that infect people, how they are spread and how they can be mitigated.

Unit 6 Remix Project and Exam	**Time Frame:** 2-5 hours
Project Goal: Students will use the skills and concepts they learned in Mission 14 and Mission 15 to create their own project. **Project Outline:** Follow the five-steps of the design process (document on next page) to design a remix project.	**Exam Goal:** Students will show mastery of Python and robotics concepts by obtaining at least 75% correct on a multiple choice test. **Project Outline:** Review concepts through additional practice and Kahoots. Then students take the exam. An option for Microsoft Forms is available. Vocabulary and concepts are two different tests.
Remix Project Assessment Opportunities ● Unit 6 Remix Project Planning Guide ● Peer reviews / Gallery walk ● Unit 6 Remix Project Rubric Checklist ● Submit Unit 6 Remix Project Program	**Unit 5 Exam Opportunities** ● Unit 6 Vocabulary review Kahoot ● Unit 6 Concepts and coding review Kahoot ● Unit 6 Vocabulary test (MS Form) ● Unit 6 Concepts & coding test (MS Form)

Supplementary Materials (available at learn.firialabs.com)

- Unit 6 Remix Project Planning Guide
- Unit 6 review and test questions
- Code Tracing Chart

Remix Ideas:

- Pick an extension idea from Mission 14 or Mission 15.
- Write a program that uses files for a different application. For example, the program could collect data from a sensor and save it to a file, and then manipulate the data, generate a bar chart, etc..
- Combine the jukebox with a file system so that you can add songs or remove songs from a list. Include CodeBot with LEDs and dance moves.
- Recreate the jukebox program, but count each time the button is pressed. Use modulo to determine which song is played. Include a file system with additional file operations. Use the other button to select a random song from the playlist.
- Think of your own creative project with line or proximity sensors, movement, LEDs and button input.

Rubric Checklist:

☐ Filename is descriptive
☐ Uses global and local variables appropriately
☐ Opens and reads from a file
☐ Iterates over a list or dictionary created from a file’s data
☐ Uses the data from a file for an application
☐ Writes to a file
☐ Controls one or more peripherals: LEDs, sound, motors
☐ Receives input from the user (button press or through the console)
☐ Defines and uses at least one function
☐ Includes something extra (sound, more than one sensor, more than one function, both buttons, etc.)
☐ Code follows programming conventions of comments, readability, indenting, and capitalization
☐ Code runs with no errors

Final Project

Time Frame: 4-8 hours

Project Goal: Students will use the skills and concepts they learned during the mission pack “Level-1 with Virtual Robotics” to create their own project.

Project Outline: Follow the five-steps of the design process (document on next page) to design a final project.

Assessment Opportunities

- Final project planning guide
- Final project rubric (included with planning guide)
- Peer reviews / Gallery walk
- Submit Final Program
- Student Reflection
- Project presentation or report

Supplementary Materials (available at learn.firialabs.com)

- Final Project Planning Guide and Rubric
- Code Tracing Chart

Remix Ideas:

- Think of your own creative project that combines concepts and code from several of the missions into something unique and that you find interesting.

Teacher Notes:

- 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.
    - ***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.
    - 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.

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.

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 student license is needed to access the Virtual Robotics curriculum**.

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.

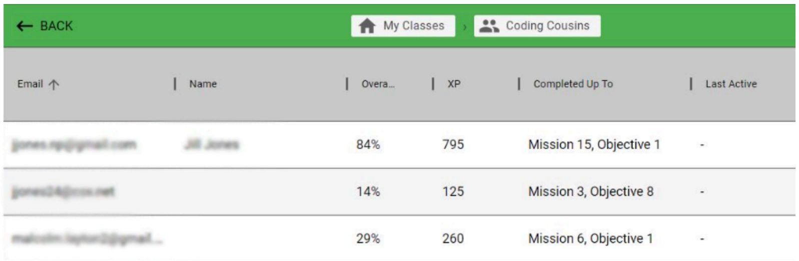
For help and step-by-step instructions, visit: https://learn.firialabs.com/curricula/code-space
If you are a **Google Classroom** teacher, you can import assignments from CodeSpace into your classes. For instructions, go to "License & Dashboard" in the Teacher Resources for Virtual Robotics.

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

Here are the basics of the CodeSpace Teacher Dashboard

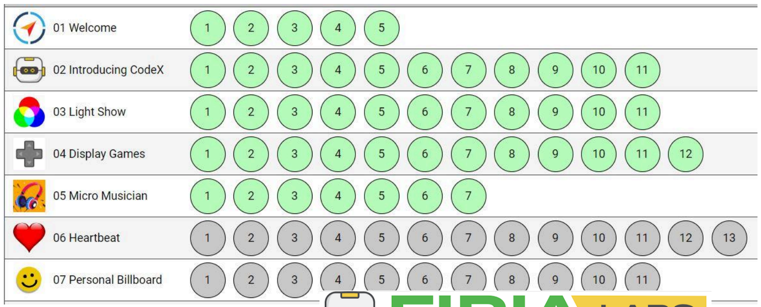
- Log in to CodeSpace and from HELP, select CLASS DASHBOARD
- Once you are in the dashboard, click + in the green bar, top right corner, to add a class.
- Assign each class a name, and allow members to join with a join code.
- You can assign Google Classroom as your LMS.
- After the class is created, 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



Email ↑	Name	Overa...	XP	Completed Up To	Last Active
[redacted]	[redacted]	84%	795	Mission 15, Objective 1	-
[redacted]		14%	125	Mission 3, Objective 8	-
[redacted]		29%	260	Mission 6, Objective 1	-

Individual progress



01 Welcome	1	2	3	4	5								
02 Introducing CodeX	1	2	3	4	5	6	7	8	9	10	11		
03 Light Show	1	2	3	4	5	6	7	8	9	10	11		
04 Display Games	1	2	3	4	5	6	7	8	9	10	11	12	
05 Micro Musician	1	2	3	4	5	6	7						
06 Heartbeat	1	2	3	4	5	6	7	8	9	10	11	12	13
07 Personal Billboard	1	2	3	4	5	6	7	8	9	10	11		

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

Each remix project has a checklist of requirements that should be met. In addition, two project rubrics are provided at learn.firialabs.com – a rubric that maps to CSTA standards (scroll to "rubrics") and a generic project rubric with a sliding scale for mastery. They can be used as a written form, or made into a digital form. The rubrics can be modified and edited 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 checklist or 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 them 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 that 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 checklist or 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 remix log for every remix.

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.

Appendix E: Python Certification

The Python Level-1 with Virtual Robotics curriculum is a pathway to top-tier certification and hands-on coding experience with a cutting edge simulation environment. The Python Level-1 with Virtual Robotics curriculum is aligned with Certiport and PCEP Certifications.

What the course offers

- **Certification Ready:** Our curriculum is tailored to help you achieve prestigious certifications like the Certiport IT Specialist-Python test and PCEP: Certified Entry-Level Python Programmer exam.
- **State-of-the-Art Simulation:** Students can experience coding like never before with our advanced virtual robotics simulation environment. They can write code, test solutions, and see their virtual robot execute commands in a realistic and interactive setting. It's coding education redefined!
- **Comprehensive Learning Path:** From foundational Python concepts to advanced applications, our curriculum provides a thorough and structured learning journey, ensuring your students are well-prepared for certification and beyond.
- **Practical Skills for the Real World:** Students gain hands-on experience that goes beyond theory. Our simulation environment bridges the gap between coding and real-world application, giving them the skills to solve problems and think critically.
- **Flexible and Accessible Learning:** With our online platform, students can access course materials, tutorials and assessments anytime, anywhere. Fit learning into their life and let them progress at their own pace, all while preparing for their certification exams.

Certifications offered

- **Certiport: Information Technology Specialist-Python Certification**
  The Information Technology Specialist program is a way for candidates to validate foundational IT skills sought after by employers. The IT Specialist program is aimed at individuals who are considering or just beginning a path to a career in information technology. Learn more about Certiport Certification.
- **PCEP: Certified Entry-Level Python Programmer**
  The PCEP-30-02 exam is a professional credential that measures the candidate's ability to accomplish coding tasks related to the essentials of programming in the Python language. A test candidate should demonstrate sufficient knowledge of the universal concepts of computer programming, the syntax and semantics of the Python language, as well as the skills in resolving typical implementation challenges with the help of the Python Standard Library. Learn more about PCEP Certification.

Crosswalk of Standards and Objectives

To see how Python Level-1 with Virtual Robotics prepares students for specific standards and objectives, go to learn.firialabs.com certification and exams.

Appendix F: Links to teacher materials

Code Solutions	Teacher Resources for Level-1 with VR (Answer Keys)
Virtual Robotics Pacing Guide	– coming soon –
Vocabulary by Mission	Teacher Resources for Level-1 with VR (General Resources)
Python Code by Mission	Teacher Resources for Level-1 with VR (General Resources)
Virtual Robotics Code Tracing Chart	Teacher Resources for Level-1 with VR (each Unit)
Unit 1 (Mission 1, 2, 3 & 4)	
Mission 1 CodeSpace Check	Teacher Resources for Level-1 with VR (Unit 1)
Mission 2 CodeBot Check	Teacher Resources for Level-1 with VR (Unit 1)
Mission 2 Review Kahoot	– coming soon –
Mission 3 Review Kahoot	– coming soon –
Mission 4 Obj 3 Planning Guide	Teacher Resources for Level-1 with VR (Unit 1)
Mission 4 Review Kahoot	– coming soon –
Unit 1 Remix Project Planning Guide	Teacher Resources for Level-1 with VR (Unit 1)
Unit 1 Vocabulary Review Kahoot	– coming soon –
Unit 1 Concepts and Coding Review Kahoot	– coming soon –
Unit 1 Vocabulary Test	Microsoft Forms (make a duplicate) – coming soon –
Unit 1 Coding and Concepts Test	Microsoft Forms (make a duplicate) – coming soon –
All Review & Test Questions	– coming soon –
Unit 2 (Mission 5-6)	
Mission 5 Review Kahoot	– coming soon –
Mission 6 Review Kahoot	– coming soon –
Unit 2 Remix Project Planning Guide	Teacher Resources for Level-1 with VR (Unit 2)
Unit 2 Vocabulary Review Kahoot	– coming soon –
Unit 2 Code and Concepts Review	– coming soon –
Unit 2 Vocabulary Test	Microsoft Forms (make a duplicate) – coming soon –
Unit 2 Coding and Concepts Test	Microsoft Forms (make a duplicate) – coming soon –
All Review & Test Questions	– coming soon –

Unit 3 (Missions 7-9)	
Mission 7 Obj. 2 Line Sensor Readings	Teacher Resources for Level-1 with VR (Unit 3)
Mission 7 Review Kahoot	– coming soon –
Mission 8 Review Kahoot	– coming soon –
Mission 9 Obj 5 Line Sensor Readings	Teacher Resources for Level-1 with VR (Unit 3)
Mission 9 Review Kahoot	– coming soon –
Unit 3 Remix Project Planning Guide	Teacher Resources for Level-1 with VR (Unit 3)
Unit 3 Vocabulary Review Kahoot	– coming soon –
Unit 3 Code and Concepts Review	– coming soon –
Unit 3 Vocabulary Test	Microsoft Forms (make a duplicate) – coming soon –
Unit 3 Coding and Concepts Test	Microsoft Forms (make a duplicate) – coming soon –
All Review & Test Questions	– coming soon –
Unit 4 (Mission 10-11)	
Mission 10 Review Kahoot	– coming soon –
Mission 11 Review Kahoot	– coming soon –
Unit 4 Remix Project Planning Guide	Teacher Resources for Level-1 with VR (Unit 4)
Unit 4 Vocabulary Review Kahoot	– coming soon –
Unit 4 Coding and Concepts Review	– coming soon –
Unit 4 Vocabulary Test	Microsoft Forms (make a duplicate) – coming soon –
Unit 4 Coding and Concepts Test	Microsoft Forms (make a duplicate) – coming soon –
All Review & Test Questions	– coming soon –
Unit 5 (Mission 12-13)	
Mission 12 Review Kahoot	– coming soon –
Mission 13 Review Kahoot	– coming soon –
Unit 5 Remix Project Planning Guide	Teacher Resources for Level-1 with VR (Unit 5)
Unit 5 Vocabulary Review Kahoot	– coming soon –
Unit 5 Coding and Concepts Review	– coming soon –
Unit 5 Vocabulary Test	Microsoft Forms (make a duplicate) – coming soon –
Unit 5 Coding and Concepts Test	Microsoft Forms (make a duplicate) – coming soon –

All Review & Test Questions	– coming soon –
Unit 6 (Mission 14-15)	
Mission 14 Review Kahoot	– coming soon –
Mission 15 Review Kahoot	– coming soon –
Unit 6 Remix Project Planning Guide	Teacher Resources for Level-1 with VR (Unit 6)
Unit 6 Vocabulary Review Kahoot	– coming soon –
Unit 6 Coding and Concepts Review	– coming soon –
Unit 6 Vocabulary Test	Microsoft Forms (make a duplicate) – coming soon –
Unit 6 Coding and Concepts Test	Microsoft Forms (make a duplicate) – coming soon –
All Review & Test Questions	– coming soon –
Final Project	
Final Project Planning Guide	Teacher Resources for Level-1 with VR (Assessments)