Level-1 Python with Virtual Robotics

CodeSpace Mission Pack

Teacher's Manual

Table of Contents

Introduction

Welcome!
This guide book will give you everything you need to make the most of the Firia Labs Python with Robots Coding Kits.

For many students and teachers, this is their very first exposure to text-based coding. If that's your situation, don't worry! We've designed the kits and this manual to gently guide you from "absolute beginner" to a very comfortable level of proficiency.

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.

Email us at: support@firialabs.com

If there's a problem that needs our attention, you can create a support ticket and we'll get back to you directly! You'll also find a community forum on our "On Fire With Firia" Facebook page, where you can ask questions and post ideas, or share your latest projects with other CodeSpace users!

Our Approach

Project Based Motivation

Student: *"Why are we even learning this?"*

Sound familiar? 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 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!

We have also thrown a few "gamification" elements, such as Experience Points (XP), into our approach to provide additional motivators. But we like to remind students: it's not about "points" - it's about "projects"!

Type it In

Student: *"Hey, I can't copy and paste the code from the lesson examples!"*

Prior to our extensive testing of this program on groups of 4th—12th graders, we were concerned that the "typing burden" might be a problem. But we were willing to risk it, because:

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

The last point above is crucial. 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! We list some ideas for re-mixing each project's concepts later in this guide.

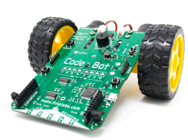
Remember that students are learning programming skills they could use to build *any* application—from controlling a rocket-ship to choreographing dance moves. Nurture the creativity, explore, and instill the Joy of Coding!

How is this different?

There are so many approaches to teaching coding. How is this different?

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

- ➢ Teaches a real, professional programming language. Even younger students appreciate that you can make real money with these exact skills.
- ➢ Gives students the tools to create *anything* they can imagine. Beyond the projects and curriculum, we give students a full-fledged software development environment. These are professional-strength tools for writing code. *(Contrast that with other approaches that only provide a game-playing environment. Once you "win", then what?)*

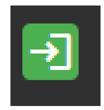
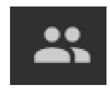


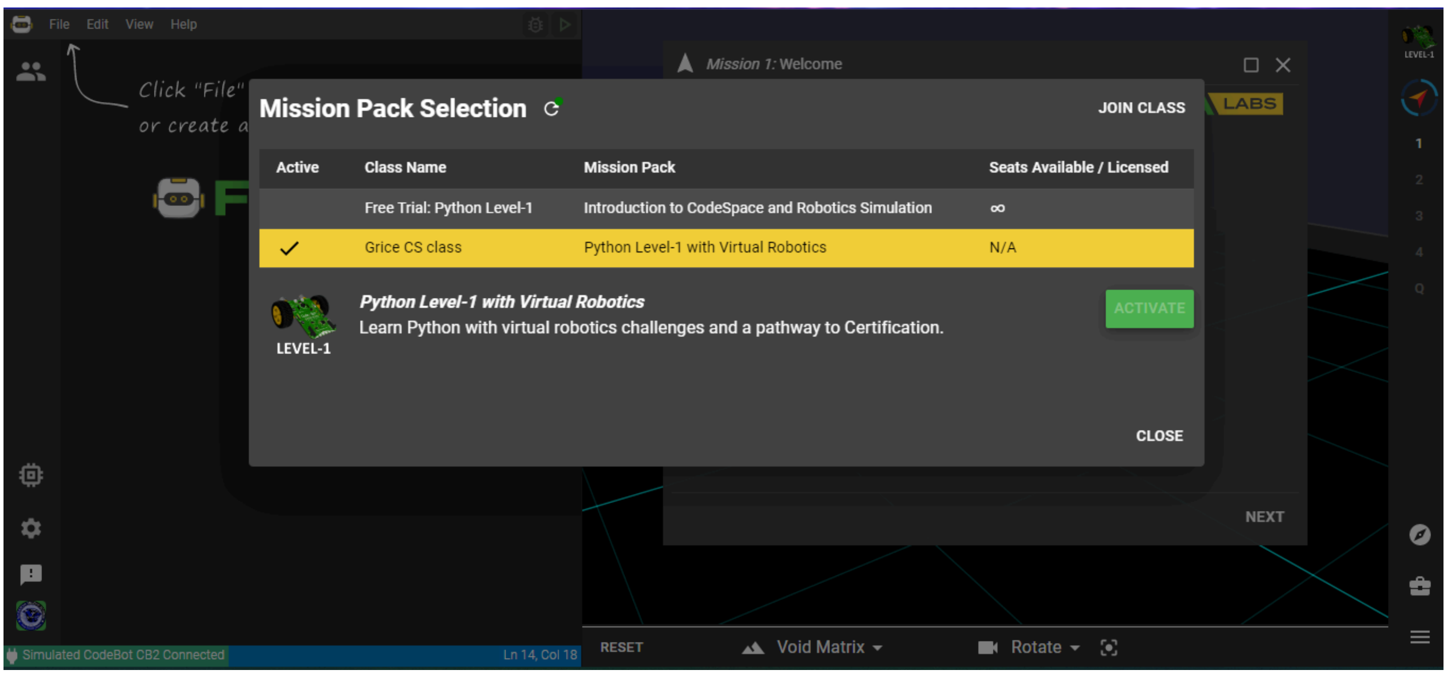
CodeSpace Overview

The CodeSpace Web Application

Ready to Code? We've made it really easy to get started!

Here are the basic steps:

1. Open your Chrome web browser
2. Go to https://sim.firialabs.com
3. Login to your account or create one (click in the bottom left corner) 
4. Select Class (click the two people icon in the upper left corner) 



Troubleshooting

Help! It's not working!?

What about problems with logging in, python coding, or other issues?

For coding problems, the first thing to try is to go back to the simplest example that *does* work for you. If there are error messages you don't understand, let us know about them. For that and any other issues, file a ticket at the support link above, go to our "On Fire With Firia" Facebook page and post the question, or email us at support@firialabs.com. We have real humans eager to help solve your problems!

Helpful Hints

Appendix A: Mission and Objective Contents, including all CodeTreks and Solutions

Appendix B: The Toolbox - all tools revealed!

Classroom Preparation

One to One

Writing code is similar in many ways to literary writing. There are grammar and syntax rules that must be followed, all while composing a meaningful narrative to satisfy the writer's objectives.

Just as developing writing skills requires individual practice, learning to code requires that students compose and test their work individually. They need to make their own mistakes, and struggle through correcting them.

Pacing and Remixing

We suggest that students be allowed a minimum of 30-minutes per session, at least until they get through the first two projects. In our experience, many students will stay engaged in excess of 90 minutes of one-on-one time working through projects. Of course, this depends on the students and the dynamics of the particular classroom. There's no substitute for a teacher's understanding of what works for a particular group of students. Experiment, and find what works for you! In the pacing guidelines below, the suggested days are based on a 90 minute block. Adjust accordingly to your school day. Because of the time it takes to set up and tear down, it may take *more than* twice as many days in a 45 minute period.

Naturally, students will progress at different speeds. Since the material is set up for independent study, you have the *option* of letting faster students move ahead to more advanced projects independently.

Remixing provides an alternative that can keep groups more synchronized in their progress through the projects. Each project can be modified, extended, and enhanced. Many students will want to experiment with what they've learned, and we offer suggestions along the way to spur this creative tinkering. If you want to keep a groups' progress in sync, instruct accelerated students to *remix* the current project upon completion, rather than moving to the next one.

We want the teachers to feel free to remix too! Create your own lesson plans using the same template as below. Then share your ideas with our online PLN at our facebook page On Fire With Firia!

Assessing Student Created Project Remixes

We recommend, in order to generate mastery, a student should practice what they are learning. One way to do this is to create a remix of each mission. A generic project rubric for these remixes can be found here as either a printable version or a Google form for paperless grading. The rubric is intended to be used for any Codespace project, but *not all standards are met with every project.* Make a copy and edit as needed. You may also want to add custom requirements or point values specific for your class. A project planning sheet is also available on the support page. Students should create a plan (and perhaps get it approved by the teacher) before they begin. 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. An example flowchart is available for your guidance when teaching students how to make a flowchart of their ideas before they begin coding. Technokids explains flowcharts with more examples and a video at the end.

Students should receive a copy of the rubric before beginning a project. You may want to make copies for all students at the beginning of the course to put in their class notebooks, and then post specific project rubrics electronically as you start a new unit. Discuss the criteria and what it means to earn mastery. It is beneficial to give students time to revise and improve upon their projects (as time permits). Students who simply achieve "Proficient" may be motivated to earn "Mastery," so decide what your classroom policy and expectations will be and explain it to students early on. You may need to revise policies as you get to know your students and observe how CodeSpace works for them, so flexibility is important!

Student-Teacher and peer conferencing are 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. Before students submit, they should complete a peer review. This may take modeling a few times before students do it correctly. They should go through the rubric and test the program just as you would. This will give them the chance to find and correct mistakes before doing a student-teacher conference. Once they submit, call students up for a conference. Share the Google Forms version of the rubric (note-remember to edit as you did for the rubric you distributed at the beginning of the project) with your students. Begin with an open-ended question, such as, "Tell me about your project," before moving 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 ask for evidence as to why. Students are often much more critical of their work than need be. It's a good time to emphasize challenges and mistakes as learning opportunities, rather than just being "wrong." If there is room for improvement and still time in the submission window, students should be allowed to debug and improve before submitting.

Students who are finished may enjoy having time to work on other unscripted projects while they wait for their classmates to finish conferencing. Again, this is not wasted time! Learning through trial and error is time well-spent.

A second way to assess students is to have them take practice certification tests. The students and teachers will see just how much the students are learning by charting their scores before starting the modules, after each module and after they have completed all of the modules. These modules are created to teach all concepts needed in order to pass either the Certiport IT Specialist-Python or OpenEDG - PCEP certifications.

The next few pages discuss these certification pathways for Python and how Level 1 Python with Virtual Robotics is aligned with these standards.

Testing Services

Certiport IT Specialist - Python

Background

According to the Certiport website, "The Information Technology Specialist program is a way for students to validate entry level IT skills sought after by employers. The IT Specialist program is aimed at candidates who are considering or just beginning a path to a career in information technology. Students can certify their knowledge in a broad range of IT topics, including software development, database administration, networking and security, mobility and device management, and coding." Python is one of the coding language pathways. "Candidates for this exam will demonstrate that they can recognize, write, and debug Python code that will logically solve a problem."

Test Format and Administration

This is a computer based, online, 50 minute exam with 33-43 questions.

Practice Materials

Certiport offers CertPREP practice tests, powered by GMetrix, cost$

Content Overview

Certiport IT Specialist Exam Objectives - Python

OpenEDG PCEP

Background

OpenEDG offers a sequence of Python certifications.

Test Format and Administration

- PCEP-30-02 – Exam: 40 minutes, NDA/Tutorial: 5 minutes
- PCEP-30-01 – Exam: 45 minutes, NDA/Tutorial: 5 minutes
- 30 Questions each
- Single- and multiple-select questions, drag & drop, gap fill, sort, code fill, code insertion | Python 3.x

Practice Materials

Python Essentials lessons through PCEP

Content Overview

PCEP Certified Entry Level Python Programmer Exam Syllabus EXAM PCEP-30-02 - Active

Firia Labs Python Level-1 Learning Objectives

This is a unified set of Learning Objectives covering the requirements of both Certiport and OpenEDG.

Ref	Category	Concept	Focus Mission	Other Missions
1.1	builtins	input()	Line Sensors	Scoreboard
1.2	builtins	len()	Robot Metronome	Scoreboard
1.3	builtins	built-in functions	Dance Bot	
1.4	builtins	print() with sep, end params	Dance Bot	
2.1	concepts	Interpreter vs Compiler	Teacher Manual	
2.2	concepts	Source code vs Object (machine) code	Teacher Manual	
2.3	concepts	coding style, PEP8 basics	Teacher Manual	
2.4	concepts	Errors: Syntax, Runtime, Logic	Teacher Manual	Scoreboard
3.1	core	None	(future)	
3.2	core	identity operator: 'is'	Eternal Flame	
3.3	core	using del to "undefined" variables	(future)	
3.4	core	type inspection using type() function	Eternal Flame	
3.5	core	pass	Cyber Storm	
3.6	core	Using help() on the REPL	(future)	
3.7	core	Backslash line continuation	(future)	
3.8	core	multiple assignment (unpacking)	Music Box	
3.9	core	conditional statements: elif, else	Fido Fetch	
3.10	core	augmented assignments	Go the Distance	
3.11	core	type conversion: int()	Music Box	Rock Climber and Combo Lock
3.12	core	global vs local scope, global keyword	Line Following	
3.13	core	bool	Robot Metronome	
3.14	core	conditional statements: if	Robot Metronome	
3.15	core	Keywords vs user-defined variable names	Dance Bot	
3.16	core	Indentation	Dance Bot	

3.17	core	comments	Light the Way	
4.1	exceptions	exception handling: try, except	Scoreboard	
4.2	exceptions	exception handling: else, finally	Scoreboard	
4.3	exceptions	raising exceptions: raise	Scoreboard	
5.1	files	File I/O: append, with	Cyber Storm	
5.2	files	File existence check, deletion	Cyber Storm	
5.3	files	File I/O: open, close, read, write	Music Box	
6.1	functions	recursion	(future)	
6.2	functions	parameters vs arguments	Boundary Patrol	
6.3	functions	positional vs keyword arguments	Boundary Patrol	
6.4	functions	function return values	Line Sensors	
6.5	functions	default function parameters	Dance Bot	Boundary Patrol
6.6	functions	defining functions	Dance Bot	
7.1	loops	continue	(future)	
7.2	loops	while-else, for-else	(future)	
7.3	loops	using for loop to iterate over string	(future)	
7.4	loops	multiple assignment in for loop	Music Box	
7.5	loops	using for loop to iterate over list	Music Box	
7.6	loops	break	Line Sensors	Cyber Storm
7.7	loops	while loop	Dance Bot	
7.8	loops	for loop, range()	Dance Bot	
8.1	math	float (type and coercion/ctor)	Eternal Flame	
8.2	math	Scientific notation	(future)	
8.3	math	bitwise operators: ~	(future)	
8.4	math	bitwise operators: &	Combination Lock	
8.5	math	bitwise operators: |	Combination Lock	Scoreboard
8.6	math	bitwise operators: ^	Combination Lock	
8.7	math	int (type and coercion/ctor)	Music Box	
8.8	math	Modulo %	Runway Ops	

8.9	math	Numeric multiply * operator	Go the Distance	
8.10	math	Numeric divide / operator	Go the Distance	
8.11	math	Integer division //	Go the Distance	Runway Ops
8.12	math	hex and octal literals	Combination Lock	
8.13	math	Power ** operator	Combination Lock	Runway Ops
8.14	math	boolean 'and'	Line Following	
8.15	math	boolean 'or'	Line Following	
8.16	math	operator precedence	Robot Metronome	
8.17	math	bitwise shifts: << >>	Robot Metronome	Combination Lock & Scoreboard
8.18	math	boolean 'not'	Robot Metronome	Scoreboard
8.19	math	comparison operators	Robot Metronome	
8.20	math	binary literals	Light the Way	Combination Lock
9.1	modules	datetime module (strftime, strptime)	Time Machine	
9.2	modules	math module	Rock Climber	
9.3	modules	random module	Eternal Flame	
9.4	modules	import of modules	Light the Way	
9.5	modules	Using unittest	Teacher Manual	
9.6	modules	os, sys, os.path, io	Teacher Manual	Cyber Storm
10.1	sequences	using list() constructor	Traffic Light	
10.2	sequences	using tuple() constructor	Traffic Light	
10.3	sequences	containment tests: 'in' and 'not in'	Cyber Storm	
10.4	sequences	dictionary: copy() method	Traffic Light	
10.5	sequences	list: copy() method and [:] to copy	Traffic Light	
10.6	sequences	slicing lists	Traffic Light	
10.7	sequences	copying a list	Traffic Light	
10.8	sequences	negative indices	Eternal Flame	
10.9	sequences	list: extend()	Traffic Light	
10.10	sequences	list operator: +	Traffic Light	
10.11	sequences	list operator: *	Runway Ops	

10.12	sequences	list: insert()	Traffic Light	
10.13	sequences	list: remove()	Traffic Light	
10.14	sequences	list: del (index or slice)	Traffic Light	
10.15	sequences	list/tuple: index()	Traffic Light	
10.16	sequences	list/tuple: sorted(), reversed()	Eternal Flame	
10.17	sequences	dictionary: keys(), items(), values()	(future)	
10.18	sequences	list: sort()	Eternal Flame	
10.19	sequences	list: append()	Music Box	
10.20	sequences	using dict() constructor	(future)	
10.21	sequences	tuple: literals and usage	Line Following	
10.22	sequences	dictionary: literals and usage	Line Following	
10.23	sequences	list comprehensions	Line Following	
10.24	sequences	Nested lists/tuples: matrices	Line Sensors	
10.25	sequences	list: literals and usage	Robot Metronome	
11.1	strings	string (type and coercion/ctor)	Cyber Storm	
11.2	strings	slicing strings	Cyber Storm	
11.3	strings	string escape sequences	Cyber Storm	
11.4	strings	multiline strings	Music Box	
11.5	strings	string formatting with f-strings	Go the Distance	
11.6	strings	string operator: +	Scoreboard	Cyber Storm
11.7	strings	string operator: *	Rock Climber	
11.8	strings	type conversion: str()	Time Machine	Combination Lock
11.9	strings	string formatting with string.format()	Rock Climber	
12.1	tools	docstrings	Boundary Patrol	
12.2	tools	Using pydoc	Teacher Manual	

**** Any mission referred to in the above table that you do not currently see on sim.firialabs.com is coming soon. ****

Level 1 Python with Virtual Robots Unit Overview

Unit 0: Coding Unplugged (5-10 days*)

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 here.

Unit 1: Introductory Missions (7 days*)

Students will learn the basics of coding in Python and the CodeBots LEDs, and motors.

Mission 1: Welcome

Mission 2: Introducing CodeBot

Mission 3: Light the Way

Mission 4: Get Moving

Unit 2: Inputs and Outputs (10 days*)

Students will learn how to use the CodeBot LEDs, Buttons, speakers and motors.

Mission 5: Dance Bot

Mission 6: Robot Metronome

Unit 3: Get Moving (15 days*)

Students will learn how to optimize the CodeBot sensors and motors.

Mission 7: Line Sensors

Mission 8: Boundary Patrol

Mission 9: Line Following

Unit 4: Synthesize (15 days*)

Students will learn how to use sensor data and botservices to synthesize all you've learned!

Mission 10: Fido Fetch

Mission 11: Airfield Ops

Mission 12: King of the Hill

Mission 13: Going the Distance

Mission 14: Music Box

Mission 15: Cyber Storm

Note* In the pacing guidelines, the suggested days are based on a 90 minute block. Adjust accordingly to your school day. Because of the time it takes to set up and tear down, it may take **more than twice as many days in a 45-50 minute period. This is pacing for just the missions without remixes. Remixes would add time to this curriculum. We suggest giving at least two hours to create a well planned remix.*

<table>
<tr><th colspan="3">Level 1 Python with Virtual Robots Pacing Guide</th></tr>
<tr><td>Week 1</td><td colspan="2">First Days
Set-up, Unplugged Activities
Dedicate time to getting to know your students, assess their prior knowledge, and build a foundation of computer science basics.</td></tr>
<tr><td>Week 2</td><td>Mission 1 & 2
Welcome & Introducing CodeBot
A visual and hands-on tour of the components of your 'bot.</td><td>Mission 3 & 4
Light the Way & Get Moving
These missions take you step-by-step through coding projects involving sequences of motor control and LED lights. Learn how to turn on sound.</td></tr>
<tr><td>Week 3</td><td colspan="2">Mission 5
Dance Bot
This mission teaches you about loops, debugging, variables, functions, and algorithms.</td></tr>
<tr><td>Week 4</td><td colspan="2">Mission 6
Robot Metronome
This mission turns your CodeBot into a time-keeping device that a musician can set to the tempo of their choice.</td></tr>
<tr><td>Week 5</td><td colspan="2">Mission 7
Line Sensors
This mission uses the line sensors to navigate your CodeBot.</td></tr>
<tr><td>Week 6</td><td>Mission 8
Boundary Patrol
The mission teaches you how to program your CodeBot to roam a fenced area, using line sensors to stay in bounds.</td><td>Mission 9
Line Following
The mission has your CodeBot mastering the biggest and baddest line-course around.</td></tr>
<tr><td>Week 7</td><td colspan="2">Mission 10
Fido Fetch
The mission trains your CodeBot to fetch using a dictionary of commands.</td></tr>
<tr><td>Week 8</td><td>Mission 11
Airfield Ops
The mission teaches you some unique programming concepts to help with airfield runway operations.</td><td>Mission 12
King of the Hill
The mission teaches all about the CodeBot’s accelerometer.</td></tr>
<tr><td>Week 9</td><td>Mission 13
Going the Distance
The mission teaches about the CodeBot’s wheel encoders and all the gritty details of those glorious rotating discs.</td><td>Mission 14
Music Box
The mission turns your CodeBot into a jukebox and teaches about Python’s file operations.</td></tr>
</table>

Level 1 Python with Virtual Robots Lesson Plans		
UNIT 1: Introductory Missions	**MISSION 1 & 2: Welcome & Introducing CodeBot**	**# HOURS:** 1-2
MISSION GOALS: Students will learn about the CodeBot hardware and the simulation environment.	**DAILY MATERIALS:** ● Google Chrome	**VOCABULARY:** ● Peripherals ● CPU

FOCUS STANDARDS:

LEARNING TARGETS:

- I can navigate CodeSpace.
- I can identify the main components of the CodeBot.
- I can create a new program and write code using conventions of capitalization and punctuation specific to Python.

SUCCESS CRITERIA:

- ❑ Identify major features of the CodeSpace interface: Text Editor, Objective Panel, Mission Bar, Toolbox, XP, Simulation Toolbar, and Navigation Controls.
- ❑ Identify major parts of the CodeBot: USB connector, LEDs, Reboot button, Power switch

KEY CONCEPTS:

- Follow instructions in the **Objective panel** carefully. There is a lot of **important** reading!
- Look for "tool icons" to collect coding tools in your **Toolbox** as you go.

DISCUSS REAL WORLD APPLICATIONS:

Make sure each student takes the time to personally inspect different views of the CodeBot. Discuss the fact that all the electronic devices they use have similar circuit boards inside. The tools and techniques they're learning apply to all the electronic devices they use every day!

Challenge students to name a few devices they use every day that might contain computer chips or "microcontrollers" such as the one on the bot. How many of the following do they think of? There are so many more!

- Microwave oven
- Cell phone
- Automobile
- Watch or fitness tracker
- Video game controller
- Refrigerator
- Home thermostat
- Coffee maker
- Bread machine
- Alarm system
- Fuel pumps
- Automatic garage doors
- Electronic locks

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.

ASSESSMENT STRATEGIES:

1.4 Checkpoint - could use as an exit slip

2.5 Submit - Students label the different parts of the CodeBot.

TEACHER NOTES:

Always refer to Appendix A if you get stuck. It has the "Answer Keys" for you

2.1 Review Inputs and Outputs

Level 1 Python with Virtual Robots Lesson Plans		
UNIT 1: Introductory Missions	**MISSION 3: Light the Way**	**# HOURS: 2-3**
MISSION GOALS: Students will learn the basics of Python.	**ADDITIONAL MATERIALS:** ● Flowchart or pseudocode paper or google doc	**VOCABULARY:** ● CodeTrek ● Byte ● Debug ● Binary

FOCUS STANDARDS: 3.17, 8.20, 9.4

LEARNING TARGETS:

- I can create a new program and write code using conventions of capitalization and punctuation specific to Python.
- I can use the "Step" feature to **debug** a program.
- I can use **binary** values to animate the LEDs.
- I can use **comments** to explain my code.
- I can plan out my code in a flowchart or pseudocode before typing it.

SUCCESS CRITERIA:

❏ Make a new file, write a program, load it to the CodeBot, and run it.
❏ Use descriptive comments.
❏ Psuedocode or flowchart effectively plans out what they want their code to accomplish.

KEY CONCEPTS:

- **Import** statements let you use code from external modules or libraries.
- Computers execute code in **sequential** steps, initially starting at the top of your file and proceeding down the page.
- Built-in functions come from libraries like **`botcore`** or **`time`**. Ex: `sleep()`,`button_a.get_presses()`
- Create flowchart and or pseudocode before typing the code.

DISCUSS REAL WORLD APPLICATIONS:
You've used some fundamental computer science and robotics principles:

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

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

ASSESSMENT STRATEGIES:
3.3 Checkpoint - could use as an exit slip
3.4 Have students practice converting base 10 to binary.
3.5 Have students create a flowchart/pseudocode BEFORE typing their code.
3.5 Have students complete a remix in Explore Mode that creates a binary message flashing the lights. (Make sure they create a pseudocode first)

TEACHER NOTES: Always refer to Appendix A if you get stuck. It has the "Answer Keys" for you
3.1 Discuss import statements. When is it better to import just the library you want, versus using the wildcard * to import all?
3.2 Discuss Binary and how computers read code in binary.
3.4 To complete the challenge to turn on ALL the LEDs, type "leds." then tab or dir(leds) to see all.
3.5 Students will create a flowchart/pseudocode describing what leds will turn on and off and in what order. This should be in their coding notebooks. It will allow them to think out their code before trying to type it like a rough draft.
3.5 Have the students use the Explore Mode (especially early finishers) This is where they can complete remix programs that they create for lessons.

Level 1 Python with Virtual Robots Flowchart/ Pseudocode		
Beginning Code: Imports Write your import commands here:		
LEDs you plan to turn on	List of LEDs to turn on:	Code to turn them on:
LEDs you plan to make blink and how many times	List of LEDs that will blink:	Code you will use to make them blink:

Level 1 Python with Virtual Robots Lesson Plans		
UNIT 1: Introductory Missions	**MISSION 4: Get Moving**	**# HOURS: 2-3**
MISSION GOALS: Students will make the CodeBot touch 4 tennis balls within 30 seconds.	**ADDITIONAL MATERIALS:** ● Code Trace Chart ● Pseudocode Chart	**VOCABULARY:** ● API ● motors ● Frequency Pitches

FOCUS STANDARDS:

LEARNING TARGETS:
- I can plan out a project using a **flowchart** (or pseudocode).
- I can rotate the CodeBot by enabling the Motors and telling them how much power to apply to each motor.
- I can apply the sleep function appropriately
- I can play a pitch from the speaker

SUCCESS CRITERIA:
- ❏ Pseudocode plans out how CodeBot will touch all 4 tennis balls
- ❏ Able to make the CodeBot Rotate at different speeds.
- ❏ Able to play sounds out of the speakers.

KEY CONCEPTS:
- Making the CodeBot rotate by enabling the motors and giving different speeds.
- Making the CodeBot play sounds out of the speaker.
- Built-in functions come from libraries like **`botcore`** or **`time`**. Ex: `sleep()`,`button_a.get_presses()`

DISCUSS REAL WORLD APPLICATIONS:
You've used some fundamental computer science and robotics principles:
- Controlling LEDs and Motors with specific timing and sequencing

This code is used in cars, stage lights, roomba vacuums, and more!

ASSESSMENT STRATEGIES:
4.3 Have students talk you through the code and what each line does.
4.3 Checkpoint.
4.3 Remix- have students Try for all 4 balls in 30 seconds.
4.4 Have students use different pitches to try to recreate a simple nursery rhyme.

TEACHER NOTES:
Always refer to Appendix A if you get stuck. It has the "Answer Keys" for you
4.3 Require students to make a flowchart and go over the engineering design process.
4.3 Do a trace chart together. Talking through what works and what does not and the importance of documenting your tries (so you do not waste time trying same code twice)

Level 1 Python with Virtual Robots Mission 4.3 Pseudocode	
Beginning Code: Imports Write your import commands here:	
What order do you plan to touch the tennis balls?	
What directions will you go in order to reach the first tennis ball?	What code will you use on the motors to make the CodeBot move those directions?
What directions will you go in order to reach the second tennis ball?	What code will you use on the motors to make the CodeBot move those directions?
What directions will you go in order to reach the third tennis ball?	What code will you use on the motors to make the CodeBot move those directions?
What directions will you go in order to reach the fourth tennis ball?	What code will you use on the motors to make the CodeBot move those directions?
What other code do you think you might need in order to meet the 30 second requirement and Why?	

Code Trace Chart (To Document how you fixed Errors)

Write this for every attempt you try so you have documentation of what each attempt did.

Code	Wanted Outcome	Correct?	Fix

Level 1 Python with Virtual Robots Lesson Plans		
UNIT 2 : Inputs and Outputs	**MISSION 5: Dance Bot**	**# HOURS: 5**
MISSION GOALS: Students will gain an in-depth understanding of CodeBot's line sensors.	**ADDITIONAL MATERIALS:** ● Code Trace Chart ● Pseudocode Chart	**VOCABULARY:** ● variable ● While & for loops ● Increment

FOCUS STANDARDS: 1.3, 1.4, 3.15, 3.16, 6.5, 6.6, 7.7, 7.8

LEARNING TARGETS:

- I can plan out a project using a **flowchart** (or pseudocode).
- I can use a while True loop.
- I can increment and decrement a variable
- I can assign **data** to a **variable**.
- I can use the "Step" feature to **debug** a program.
- I can write a **function**.
- I can use `buttons.was_pressed` to control a **function.**

SUCCESS CRITERIA:

❏ Loops used correctly to Blink LED 8 times.
❏ Increments are used correctly to count number of blinks
❏ Incrementally test code.
❏ Advanced debugger and `print( )` functions are used to test different environments.

KEY CONCEPTS:

- While loops are used to execute an algorithm.
- The colon at the end of a while statement introduces a new block of code. Everything inside the block should be indented at the same level.
- Increments (and decrements) are used to make code cleaner and more efficient.
- The CodeSpace **debugger** lets you *step* through the code one line at a time to understand what the computer is doing.
- **Variables** can be defined to hold changing values.
- CodeSpace's **Debug Console** can be used to experiment with Python's `print( )` statement.
- A **function** is a named chunk of code you can run anytime just by calling its name
- Buttons presses (**Inputs**), LEDs (**Outputs**), and Speaker sounds (Outputs) are part of the **User Interface**. They allow the user to interact with the robot.

DISCUSS REAL WORLD APPLICATIONS:

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

- Reading button inputs

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

ASSESSMENT STRATEGIES:

Always remember to have students create pseudocode for any new programs they are writing and to keep copies of these in a notebook whether digital or paper based. The Code Trace sheets should be with each program as well.

5.1 Have students remix where they make different lights blink different numbers of times and for different lengths of time.
5.2 Checkpoint
5.5 Have students journal about writing this code. What they liked, didn't like, problems they ran into, how this relates to the real world.
5.6 Have students discuss indentation of their code and whether it will work if you do not indent or is it just to make it look more readable.
5.6 Checkpoint as an exit slip
5.8 Have students turn in their code BEFORE you move them to pairs or the whole group.

TEACHER NOTES:
Remind students to put comments in their code for later reference.
Always refer to Appendix A if you get stuck. It has the "Answer Keys" for you
5.1 Discuss Variables in math class and then explain how they work in programming.
5.2 discuss the print command and what exactly it is doing (you do not see it on the CodeBot anywhere).
5.2 discuss the importance of debugging programs.
5.5 Have students fill out the Code Trace Chart and discuss it with them.
5.6 discuss algorithms and functions. (use the information in the toolbox that is given and discuss in detail)
5.8 have class discussion on code they used and why. Have them start individually for a day, then in pairs, and then as a class

Level 1 Python with Virtual Robots Lesson Plans

UNIT 2 : Inputs and Outputs	**MISSION 6: Robot Metronome**	**# HOURS: 5**
MISSION GOALS: Students will use sensor inputs to program the 'bot to play different tempos.	**ADDITIONAL MATERIALS:** ● Code Trace Chart ● Pseudocode Chart	**VOCABULARY:** ● Tempo ● BPM ● Infinite loop ● Boolean

FOCUS STANDARDS: 1.2, 3.13, 3.14, 8.16, 8.17, 8.18, 8.19, 10.25

LEARNING TARGETS:

- I can plan out a project using a **flowchart** (or pseudocode).
- I can use a `while True` loop.
- I can use `buttons.was_pressed` to control a **variable.**
- Make a toggle

SUCCESS CRITERIA:

- ❏ Loops used correctly to blink LEDs at a set tempo
- ❏ Button is pressed to toggle between sound on and off
- ❏ The LEDs light up correctly based on which tempo is selected.

KEY CONCEPTS:

- Infinite while loops are used to execute an algorithm constantly.
- Buttons presses (**Inputs**), LEDs (**Outputs**), and Speaker sounds (Outputs) are part of the **User Interface**. They allow the user to interact with the robot.
- Bit-shift operator used to change which LED lights up.

DISCUSS REAL WORLD APPLICATIONS:
Musicians can keep a tempo going like a metronome.

ASSESSMENT STRATEGIES:
6.1 Exit slip on what BMP and tempo mean
6.4 Checkpoint
6.6 Journal about the difference in buttons.was_pressed and buttons.is_pressed
6.7 Discuss what toggle means on exit slip
6.8 Checkpoint and quick check on music names for the different tempos.
6.10 Submit code Trace of how fixed error you typed in 6.9

TEACHER NOTES:
Always refer to Appendix A if you get stuck. It has the "Answer Keys" for you
Require students to make a flowchart for all code and go over the engineering design process.
Submit. Students should submit their code and documentation of the engineering design process.
6.4 discuss the math behind calculating the BMP. Maybe have them calculate other beats per minute and show what sleep(#) would be used

Level 1 Python with Virtual Robots Lesson Plans		
UNIT 3: Get Moving	**MISSION 7: Line Sensors**	**# HOURS: 5**
MISSION GOALS: Students will use sensor inputs to program the 'bot to navigate around lines.	**ADDITIONAL MATERIALS:** ● Code Trace Chart ● Pseudocode Chart ● Paper to chart the table of results for each cardinal and intermediate direction.	**VOCABULARY:** ● Phototransistor ● Emitter ● Detector ● Reflector ● Reflectivity ● print( ) statement

FOCUS STANDARDS: 1.1, 6.4, 7.6, 10.24

LEARNING TARGETS:
- I can plan out a project using a **flowchart** (or pseudocode).
- I can use the Advanced Debugger to get real-time sensor values.
- I can use `ls.read` to get real-time line sensor values.

SUCCESS CRITERIA:
- ❏ Advanced debugger and `print( )` functions are used to test different environments.

KEY CONCEPTS:
- Gain an in-depth understanding of CodeBot's Line Sensor
- **Analog Sensors** are non-contact sensors used in many industrial and commercial applications.

DISCUSS REAL WORLD APPLICATIONS:
Robot Housekeepers
Self-Driving cars
All kinds of AI systems.

ASSESSMENT STRATEGIES:
7.1 Checkpoint as exit slip
7.2 Exit slip of code to spin CodeBot SLOWLY clockwise
7.2 Submit their table of data for the cardinal directions and the CodeBot's reading at each.(They will have to start and stop the program in each cardinal and intermediate direction in order to find the correct readings in the console.
7.3 Discuss Compound inequalities and how they are being used in this code
7.3 Checkpoint as exit slip
7.4 Discuss Absolute value and how it is being applied here as well as constants and global variables.
7.6 Submit code with breakdown of what each line does (must include detailed comments)
7.6 Checkpoint as exit slip

TEACHER NOTES:
Always refer to Appendix A if you get stuck. It has the "Answer Keys" for you
Require students to make a flowchart for all code and go over the engineering design process.
Submit. Students should submit their code and documentation of the engineering design process.
7.2 Discuss the difference in clockwise and counterclockwise
Discuss the terms API and REPL and make sure students understand their use across programming languages.
7.6 Discuss Matrices and how to access the different data in it.

Level 1 Python with Virtual Robots Lesson Plans

UNIT 3: Get Moving	MISSION 8: Boundary Patrol	# HOURS: 2-3
MISSION GOALS: Students will use sensor inputs to program the 'bot to roam a fenced (lined) area.	**ADDITIONAL MATERIALS:** ● Code Trace Chart ● Pseudocode Chart	**VOCABULARY:** ● Docstring (in Code Trek of 8.6) ● Constants ● Parameter vs argument

FOCUS STANDARDS: 6.2, 6.3, 12.1

LEARNING TARGETS:
- I can plan out a project using a **flowchart** (or pseudocode).
- I can display the boolean results on the LED above each line sensor.
- I can make a contact counter to show each line-detect on the user LEDs.
- I can teach the bot to stay inside the lines.
- I can use the proximity sensor `detect( )` API to make a presence detector.
- I can experiment with light and dark surfaces to find the ideal emitter power and detection threshold levels for each environment.
- I can apply previous knowledge of the motors to rotate to face an object moving in front of it.
- I can create an algorithm to track an object and chase after it.

SUCCESS CRITERIA:
- ❑ Value threshold and comparison operator are customized for the specific testing environment.
- ❑ Create a function that turns on `leds.ls_num` above each line sensor.
- ❑ Use a variable increment to count up on `leds.user` each time a line is detected.
- ❑ Reuse code from *Line Sensors* to drive bot.
- ❑ Write code that detects an object using the proximity sensors and light the LED near the corresponding sensor.

KEY CONCEPTS:
- Use **threshold comparison operations** to make decisions with sensor data.
- CodeSpace's **Debug Console** can be used to experiment with Python's `print( )` statement.
- Engineers build in safety features so the device doesn't run on startup, but will wait for the user.
- Autonomous robots use sensor data to make decisions and take action in its unique environment.
- A detection `threshold` from 0%-100% controls how much light is needed for a `True` detection. *If you decrease the* `threshold` *value, the 'bot works well even on a white surface.*

DISCUSS REAL WORLD APPLICATIONS:
- **Automatic Guided Vehicles (AGVs)** use this kind of code to zoom around warehouse distribution centers, getting packages to you!
- Robots are used to clean up environmental waste, explore underground mines, discover shipwrecks, and do other tasks deemed unsafe for humans.
- The kind of code you've written is inside stuff you might use every day, without even thinking about it! Touchless faucets, soap dispensers, and hand dryers, automatic doors, vehicle navigation and safety systems, and factory automation systems.

ASSESSMENT STRATEGIES:
8.2 Checkpoint
8.4 Submit pseudocode
8.4 Checkpoint
8.5 Have the students walk through each step of code and explain what is happening to check for understanding.

TEACHER NOTES:
Always refer to Appendix A if you get stuck. It has the "Answer Keys" for you
Require students to make a flowchart for all code and go over the engineering design process.

Submit. Students should submit their code and documentation of the engineering design process.
8.1 Remember to discuss where the code prints to (console)

UNIT 3: Get Moving	MISSION 9: Line Following	# HOURS: 2-3
MISSION GOALS: Students will use sensor inputs to program the 'bot to follow a line course.	**ADDITIONAL MATERIALS:** ● Code Trace Chart ● Pseudocode Chart ● Table for charting sensor data	**VOCABULARY:** ● REPL ● Tuple ● Algorithm ● Python Dictionary ● Global variables

FOCUS STANDARDS: 3.12, 8.14, 8.15, 10.21, 10.22, 10.23

LEARNING TARGETS:

- I can plan out a project using a **flowchart** (or pseudocode).
- I can apply previous knowledge of the motors and line sensing to make my CodeBot follow a path.
- I can use proximity sensors to avoid obstacles.
- I can use navigation code to navigate an obstacle course.

SUCCESS CRITERIA:

- ❏ My CodeBot followed a curved line and hit the target within the time frame.
- ❏ 'Bot successfully navigates obstacle course:
  - ❏ Does not veer off course
  - ❏ Stays within boundary lines
  - ❏ Finish the course within the time limit.

KEY CONCEPTS:

- Write code that uses the line sensors.
- APIs
- PIDs
- Understand and use the data from line sensors to navigate the path.
- Create a dictionary for the CodeBot to use.

DISCUSS REAL WORLD APPLICATIONS:
Self driving car

ASSESSMENT STRATEGIES:
9.2 Checkpoint as exit slip
9.4 Psuedocode and or Flowchart submitted with code tracing chart.
9.4 Checkpoint as exit slip
9.5 Have the students create a table (google sheets, excel, on paper) of the data they collect in the console and turn it in.
9.6 Checkpoint as exit slip
9.8 Students can describe global variables in an exit slip
9.8 checkpoint as exit slip

TEACHER NOTES:
Always refer to Appendix A if you get stuck. It has the "Answer Keys" for you
Require students to make a flowchart for all code and go over the engineering design process.
Submit. Students should submit their code and documentation of the engineering design process.
9.2 Talk through with students what the REPL is and why you would use it instead of typing in the text editor.
9.4 If CodeBot does not follow the line to checkpoint 2, try changing the speed and or turn radius until you find one that works.
9.6 Discuss proportions in math and how it relates to this code.
9.6 Make sure students document ALL possible choices on the last objective. They are SUPPOSED to get an error. When they go to

the next objective it will explain it.
9.7 can make you dizzy to watch so make sure epileptics are careful with this one.

UNIT 4: Synthesize	MISSION 10: Fido Fetch	# HOURS: 5
MISSION GOALS: Students will train the CodeBot to fetch using python dictionaries and console inputs.	**ADDITIONAL MATERIALS:** ● Code Trace Chart ● Pseudocode Chart	**VOCABULARY:** ● Python dictionary ● Refactoring ● KeyError

FOCUS STANDARDS: 8.4, 8.5, 8.6, 8.12, 8.13

LEARNING TARGETS:
- I can plan out a project using a **flowchart** (or pseudocode).
- Create a Python dictionary of commands for Fido.
- Use console inputs to call different commands in the dictionary.
- Refactor code to make it more readable

SUCCESS CRITERIA:
- ❏ The robot moves when say 'come' and stops when say 'stay'
- ❏ Code still works after refactoring
- ❏ Fido eats all the treats

KEY CONCEPTS:
- Add a function to your created dictionary
- Refactoring makes code easier to read and change

DISCUSS REAL WORLD APPLICATIONS:

ASSESSMENT STRATEGIES:
10.3 Have the students explain all the parts of a dictionary (the key, and value)
10.3 checkpoint as an exit slip
10.6 submit your pseudocode or flowchart
10.7 checkpoint as an exit slip

TEACHER NOTES:
Always refer to Appendix A if you get stuck. It has the "Answer Keys" for you
Require students to make a flowchart for all code and go over the engineering design process.
Submit. Students should submit their code and documentation of the engineering design process.
10.2 remember that the students must be zoomed in in order to hear Fido "speak"
10.9 discuss the value of using the arrow keys instead of retyping so much.

UNIT 4: Synthesize	MISSION 11: Airfield Ops	# HOURS: 2-3
MISSION GOALS: Students will learn some unique programming concepts that help with airfield runway operations.	**ADDITIONAL MATERIALS:** ● Code Trace Chart ● Pseudocode Chart	**VOCABULARY:** ● Integer division ● modulo

FOCUS STANDARDS: 9.2, 11.7, 11.9

LEARNING TARGETS:
- I can plan out a project using a **flowchart** (or pseudocode).
- Using line sensors to follow a line
- Turning on the speaker and LEDs
- Learn some new Python operators: `%`, `**`, and `//`

SUCCESS CRITERIA:
- ❏ My CodeBot followed the dotted line to the end
- ❏ My CodeBot stops at the end of the line
- ❏ My LEDs light up at correct positions along the airfield.

KEY CONCEPTS:
- Use python operators for division, multiplying, and remainders

DISCUSS REAL WORLD APPLICATIONS:

ASSESSMENT STRATEGIES:
11.1 submit your final code and your codetrace
11.2 submit your count so definitely have for next objective
11.6 checkpoint as exit slip

TEACHER NOTES:
Always refer to Appendix A if you get stuck. It has the “Answer Keys” for you
Require students to make a flowchart for all code and go over the engineering design process.
Submit. Students should submit their code and documentation of the engineering design process.
11.1 most students will not get first try so make them use a code trace chart and create a flowchart/pseudocode
11.4 discuss the binary representation that is in the chart
11.5 Make sure the students have their volume up and they might have to change the camera to be attached in order to be able to hear the tones.

Level 1 Python with Virtual Robots Lesson Plans		
UNIT 4: Synthesize	**MISSION 12: King of the Hill**	**# HOURS: 2-3**
MISSION GOALS: Students will use the CodeBot's accelerometer to "Master the Hill."	**ADDITIONAL MATERIALS:** ● Code Trace Chart ● Pseudocode Chart	**VOCABULARY:** ● Accelerometer ● Format specifiers ● ASCII ● Hexadecimal

FOCUS STANDARDS: 3.10, 8.9, 8.10, 8.11, 11.5

LEARNING TARGETS:

- I can plan out a project using a **flowchart** (or pseudocode).
- I can print accelerometer data to the console in 3 different ways (comma separated list, list, and tuple)
- Learn how to see the pitch and roll of the CodeBot
- Learn how to make the console show a bar graph of the information

SUCCESS CRITERIA:

- ❏ Display the Accelerometer data in three different ways (comma separated lists, lists, and tuples)
- ❏ Convert radians to degrees
- ❏ CodeBot displays the pitch and roll to the console, shows bar graph of values, and the degree measure
- ❏ CodeBot is able to drive the course Autonomously and avoids crashes.

KEY CONCEPTS:

- Understand the difference in comma separated lists, lists, and tuples
- Import the math module
- Use the 0 and 1 on keyboard to represent pushing buttons on the CodeBot to move it forward
- Understand pitch vs roll
- Full autonomy vs remote controlled
- String formatting
- Escape sequences

DISCUSS REAL WORLD APPLICATIONS:
Remote controlled cars, drones, etc.

ASSESSMENT STRATEGIES:
12.1 Checkpoint as exit slip
12.4 submit pseudocode as well as Code Trace
12.6 submit pseudocode as well as Code Trace

TEACHER NOTES:
Always refer to Appendix A if you get stuck. It has the "Answer Keys" for you
Require students to make a flowchart for all code and go over the engineering design process.
Submit. Students should submit their code and documentation of the engineering design process.
12.1 make sure on the sep= that they do not put a space between the "," and discuss the three different representations of the data
12.2 You could bring in the science teacher to discuss pitch, roll, yaw and the angles or the math teacher to speak about the geometry. Make it a cross curricular unit. The instructions in CodeTrek explain these concepts.

Level 1 Python with Virtual Robots Lesson Plans		
UNIT 4: Synthesize	**MISSION 13: Going the Distance**	**# HOURS: 2-3**
MISSION GOALS: Students will learn all about the Codebot's wheel encoders.	**ADDITIONAL MATERIALS:** ● Code Trace Chart ● Pseudocode Chart	**VOCABULARY:** ● Wheel encoder ● API function ● Circumference ● Closed loop control system vs open loop

FOCUS STANDARDS: 3.9

LEARNING TARGETS:

- I can plan out a project using a **flowchart** (or pseudocode).
- I can use the wheel encounter to calculate the distance and angle of rotation
- I can have my CodeBot Count Pulses
- I can use math to make my CodeBot move in a Arc pattern

SUCCESS CRITERIA:

- ❏ Console shows bar graph representing wheel rotation
- ❏ CodeBot moves only one full rotation
- ❏ CodeBot stops at checkpoint
- ❏ CodeBot autonomously goes around free throw ring and hits all 4 checkpoints

KEY CONCEPTS:

- Wheel encounter
- Ascii
- Bar chart
- / vs //
- Counting Pulses

DISCUSS REAL WORLD APPLICATIONS:
It's quite likely you interact with products every day which use sensors like this. From home appliances to automobiles, electromechanical systems of all kinds use sensors like this to measure rotational motion.

ASSESSMENT STRATEGIES:
13.3 submit code used in REPL
13.3 Checkpoint as exit slip
13.6 Checkpoint as exit slip with explanation for why those are the correct answers
13.8 Have students calculate 3 wheel rotations using the math on the screen
13.10 Have the students show how they calculated the speed for the formula in the code
13.10 checkpoint as an exit slip
13.11 What did they adjust the KP to and did they adjust anything else? If so, what and why?
13.12 Have students create a remix. Maybe rotate 180 degrees and go clockwise instead of counterclockwise

TEACHER NOTES:
Always refer to Appendix A if you get stuck. It has the "Answer Keys" for you
Require students to make a flowchart for all code and go over the engineering design process.
Submit. Students should submit their code and documentation of the engineering design process.
13.8 bring the math teacher in to discuss Circumference
13.8 Checkpoint moves, so have them zoom out and count the lines in order to figure out how many CM it needs to move based on number of lines between CodeBot and Checkpoint
13.12 definitely a great lesson to bring in a math guest speaker to speak about these formulas' application in the real world.

Level 1 Python with Virtual Robots Lesson Plans		
UNIT 4: Synthesize	**MISSION 14: Music Box**	**# HOURS: 2-3**
MISSION GOALS: Students will turn the CodeBot into a jukebox and learn about Python's file operations.	**ADDITIONAL MATERIALS:** ● Code Trace Chart ● Pseudocode Chart	**VOCABULARY:** ● Split function ● Frequencies ● Flushing a file ● int() function ● f.readlines()

FOCUS STANDARDS: 8.8, 10.11

LEARNING TARGETS:

- I can plan out a project using a **flowchart** (or pseudocode).
- I can play songs through its speaker using a function to translate notes and beats into frequencies
- I can store and retrieve note sequences
- I can code button pushes to change to different songs
- I can code my CodeBot to pull songs from other files in my file system

SUCCESS CRITERIA:

❏ CodeBot plays Twinkle, Twinkle, Little Star
❏ CodeBot plays Jingle Bells
❏ CodeBot code pulls information from other files in the file system you created.
❏ CodeBot plays Mary Had a Little Lamb with correct note lengths

KEY CONCEPTS:

- Create a list of the notes to play and then call each note in a for loop
- Use split function
- Write code that can read other files in the file system
- Flush a file when you want to save it but keep it open
- Add length of beat to each note
- Convert data to an integer
- Build a multidimensional list from the file_lines

DISCUSS REAL WORLD APPLICATIONS:
Jukebox

ASSESSMENT STRATEGIES:
14.2 submit code
14.3 submit code
14.9 submit your pseudocode and codetrace
14.10 Remix it for different songs and maybe make it where button 1 goes forward in the list and 0 goes backwards. Also try adding a light show to it by lighting up the leds as it plays the songs and maybe driving around while it plays.

TEACHER NOTES:
Always refer to Appendix A if you get stuck. It has the "Answer Keys" for you
Require students to make a flowchart for all code and go over the engineering design process.
Submit. Students should submit their code and documentation of the engineering design process.
*** Have students "save as" at the beginning of each new objective so that they can refer back to other methods taught ***
14.1 Bring in the music teacher to discuss and show the students the different frequencies and how they are important to music.
14.2 & 14.3 discuss other songs the students might know from their family heritage and see if you can find the notes to play on the CodeBot

14.5 This is a HUGE concept in the real world. Programmers write code all the time that pulls other files they have created.
14.5 make sure the students include the file extension in the name for the open command.

Level 1 Python with Virtual Robots Lesson Plans		
UNIT 4: Synthesize	**MISSION 15: Cyber Storm**	**# HOURS: 2-3**
MISSION GOALS: Students will help to protect an email server by using file operations.	**ADDITIONAL MATERIALS:** ● Code Trace Chart ● Pseudocode Chart	**VOCABULARY:** ●
FOCUS STANDARDS:		
LEARNING TARGETS: ● I can plan		
SUCCESS CRITERIA: ❏ CodeBot		
KEY CONCEPTS: ●		
DISCUSS REAL WORLD APPLICATIONS:		
ASSESSMENT STRATEGIES:		
TEACHER NOTES: Always refer to Appendix A if you get stuck. It has the "Answer Keys" for you Require students to make a flowchart for all code and go over the engineering design process. Submit. Students should submit their code and documentation of the engineering design process. *** Have students "save as" at the beginning of each new objective so that they can refer back to other methods taught ***		