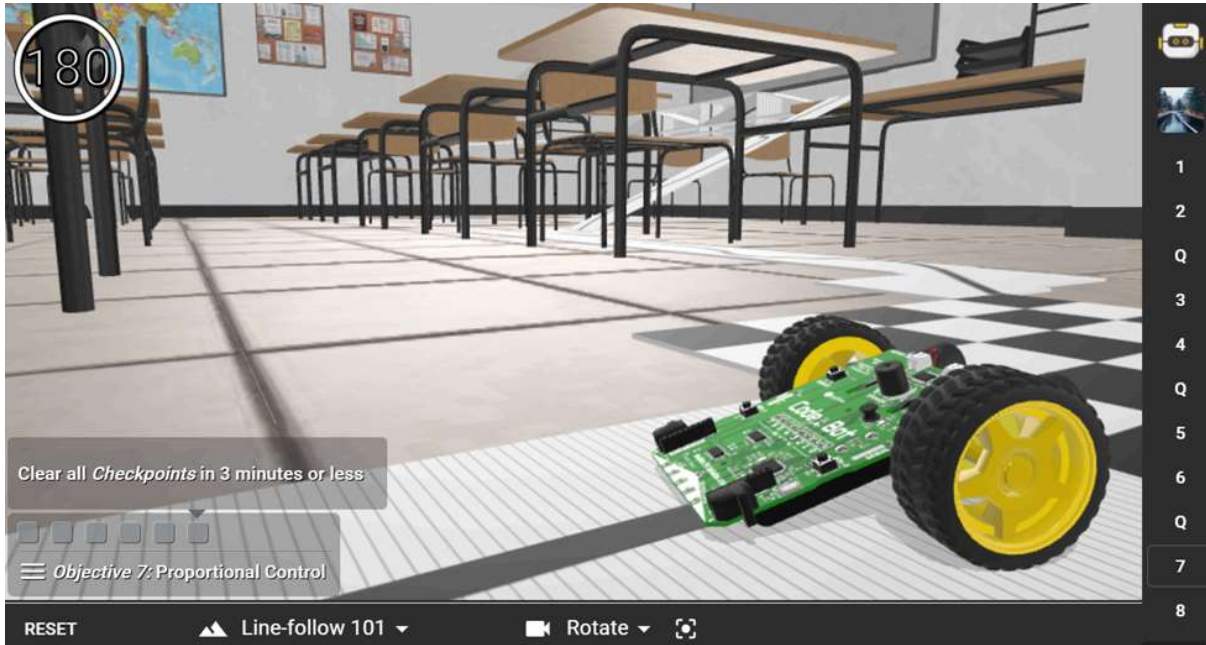




# FIRIA LABS



## Level-1 Python with Virtual Robotics

CodeSpace Mission Pack

# Teacher's Manual

# Table of Contents

Table of Contents	1
<a href="#">Introduction</a>	3
<a href="#">Our Approach</a>	4
<a href="#">How is this different?</a>	5
<a href="#">CodeSpace Overview</a>	6
<a href="#">Troubleshooting</a>	7
<b>Helpful Hints</b>	7
<b>Classroom Preparation</b>	8
<a href="#">Assessing Student Created Project Remixes</a>	9
Testing Services	10
Certiport IT Specialist - Python	10
Background	10
Test Format and Administration	10
Practice Materials	10
Content Overview	10
OpenEDG PCEP	10
Background	10
Test Format and Administration	10
Practice Materials	10
Content Overview	10
Firia Labs Python Level-1 Learning Objectives	11
<b>** Any mission referred to in the above table that you do not currently see on sim.firialabs.com is coming soon. **</b>	14
<a href="#">Level 1 Python with Virtual Robots Unit Overview</a>	15
<a href="#">MISSION 1 &amp; 2: Welcome &amp; Introducing CodeBot</a>	17
<a href="#">MISSION 3: Light the Way</a>	18
<a href="#">MISSION 4: Get Moving</a>	20
<a href="#">MISSION 5: Dance Bot</a>	23
<a href="#">MISSION 6: Robot Metronome</a>	25
<a href="#">MISSION 7: Line Sensors</a>	26
<a href="#">MISSION 8: Boundary Patrol</a>	27
<a href="#">MISSION 9: Line Following</a>	29



<a href="#">MISSION 10: Fido Fetch</a>	<a href="#">31</a>
<a href="#">MISSION 11: Airfield Ops</a>	<a href="#">32</a>
<a href="#">MISSION 12: King of the Hill</a>	<a href="#">33</a>
<a href="#">MISSION 13: Going the Distance</a>	<a href="#">34</a>
<a href="#">MISSION 14: Music Box</a>	<a href="#">35</a>
<a href="#">MISSION 15: Cyber Storm</a>	<a href="#">37</a>

## 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](mailto: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?)*





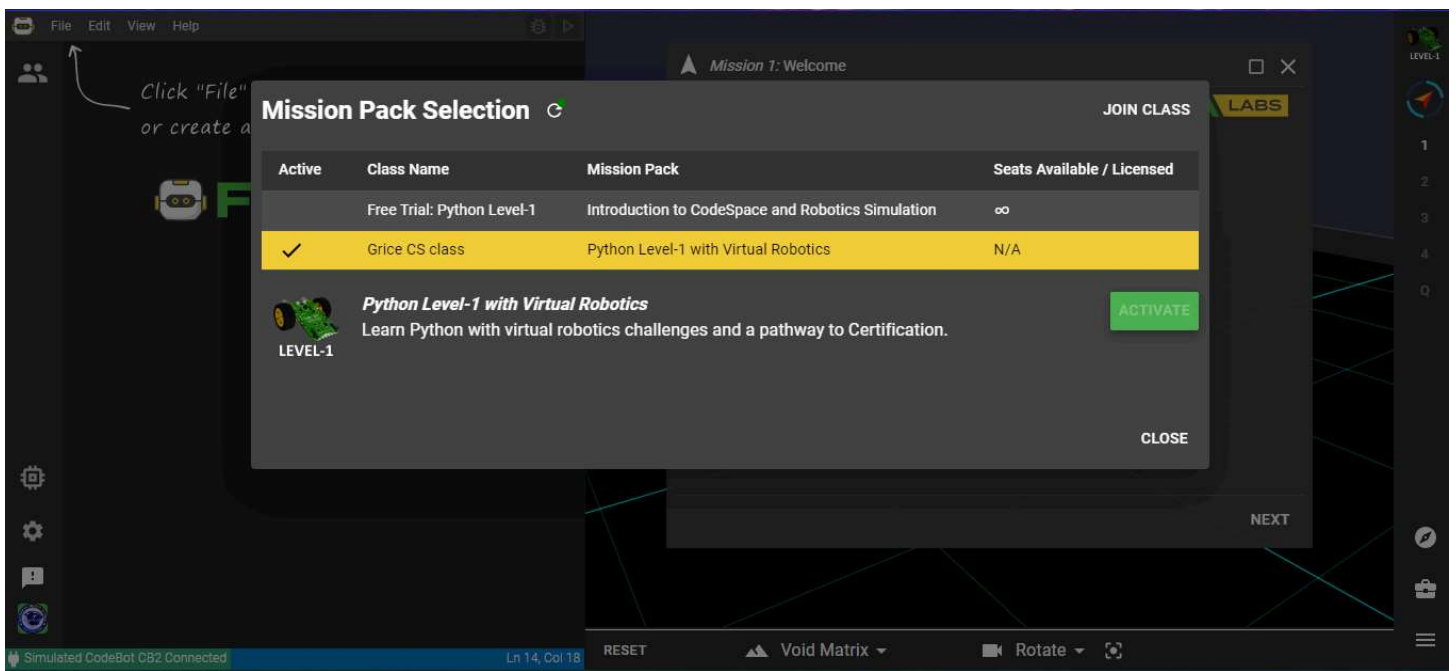
# CodeSpace Overview

## 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](mailto: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](http://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.*





## Level 1 Python with Virtual Robots Pacing Guide

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

## Level 1 Python with Virtual Robots Lesson Plans

<b>UNIT 1: Introductory Missions</b>	<b>MISSION 1 &amp; 2: Welcome &amp; Introducing CodeBot</b>	<b># HOURS:</b> 1-2			
<b>MISSION GOALS:</b> Students will learn about the CodeBot hardware and the simulation environment.	<b>DAILY MATERIALS:</b> <ul style="list-style-type: none"> <li>● Google Chrome</li> </ul>	<b>VOCABULARY:</b> <ul style="list-style-type: none"> <li>● Peripherals</li> <li>● CPU</li> </ul>			
<b>FOCUS STANDARDS:</b>					
<b>LEARNING TARGETS:</b> <ul style="list-style-type: none"> <li>● I can navigate CodeSpace.</li> <li>● I can identify the main components of the CodeBot.</li> <li>● I can create a new program and write code using conventions of capitalization and punctuation specific to Python.</li> </ul>					
<b>SUCCESS CRITERIA:</b> <ul style="list-style-type: none"> <li><input type="checkbox"/> Identify major features of the CodeSpace interface: Text Editor, Objective Panel, Mission Bar, Toolbox, XP, Simulation Toolbar, and Navigation Controls.</li> <li><input type="checkbox"/> Identify major parts of the CodeBot: USB connector, LEDs, Reboot button, Power switch</li> </ul>					
<b>KEY CONCEPTS:</b> <ul style="list-style-type: none"> <li>● Follow instructions in the <b>Objective panel</b> carefully. There is a lot of <b>important</b> reading!</li> <li>● Look for “tool icons” to collect coding tools in your <b>Toolbox</b> as you go.</li> </ul>					
<b>DISCUSS REAL WORLD APPLICATIONS:</b> <p>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!</p> <p>Challenge students to name a few devices they use every day that might contain computer chips or “microcontrollers” such as the one on the bot. How many of the following do they think of? There are so many more!</p> <table style="width: 100%; border: none;"> <tr> <td style="width: 33%; vertical-align: top;"> <ul style="list-style-type: none"> <li>● Microwave oven</li> <li>● Cell phone</li> <li>● Automobile</li> <li>● Watch or fitness tracker</li> </ul> </td> <td style="width: 33%; vertical-align: top;"> <ul style="list-style-type: none"> <li>● Video game controller</li> <li>● Refrigerator</li> <li>● Home thermostat</li> <li>● Coffee maker</li> </ul> </td> <td style="width: 33%; vertical-align: top;"> <ul style="list-style-type: none"> <li>● Bread machine</li> <li>● Alarm system</li> <li>● Fuel pumps</li> <li>● Automatic garage doors</li> <li>● Electronic locks</li> </ul> </td> </tr> </table> <p>Challenge students to describe how our lives are impacted by the above technology, and to compare how related tasks were done before computer technology was invented.</p>			<ul style="list-style-type: none"> <li>● Microwave oven</li> <li>● Cell phone</li> <li>● Automobile</li> <li>● Watch or fitness tracker</li> </ul>	<ul style="list-style-type: none"> <li>● Video game controller</li> <li>● Refrigerator</li> <li>● Home thermostat</li> <li>● Coffee maker</li> </ul>	<ul style="list-style-type: none"> <li>● Bread machine</li> <li>● Alarm system</li> <li>● Fuel pumps</li> <li>● Automatic garage doors</li> <li>● Electronic locks</li> </ul>
<ul style="list-style-type: none"> <li>● Microwave oven</li> <li>● Cell phone</li> <li>● Automobile</li> <li>● Watch or fitness tracker</li> </ul>	<ul style="list-style-type: none"> <li>● Video game controller</li> <li>● Refrigerator</li> <li>● Home thermostat</li> <li>● Coffee maker</li> </ul>	<ul style="list-style-type: none"> <li>● Bread machine</li> <li>● Alarm system</li> <li>● Fuel pumps</li> <li>● Automatic garage doors</li> <li>● Electronic locks</li> </ul>			
<b>ASSESSMENT STRATEGIES:</b> <p>1.4 Checkpoint - could use as an exit slip</p> <p>2.5 Submit - Students label the different parts of the CodeBot.</p>					
<b>TEACHER NOTES:</b> <p>Always refer to Appendix A if you get stuck. It has the “Answer Keys” for you</p> <p>2.1 Review Inputs and Outputs</p>					