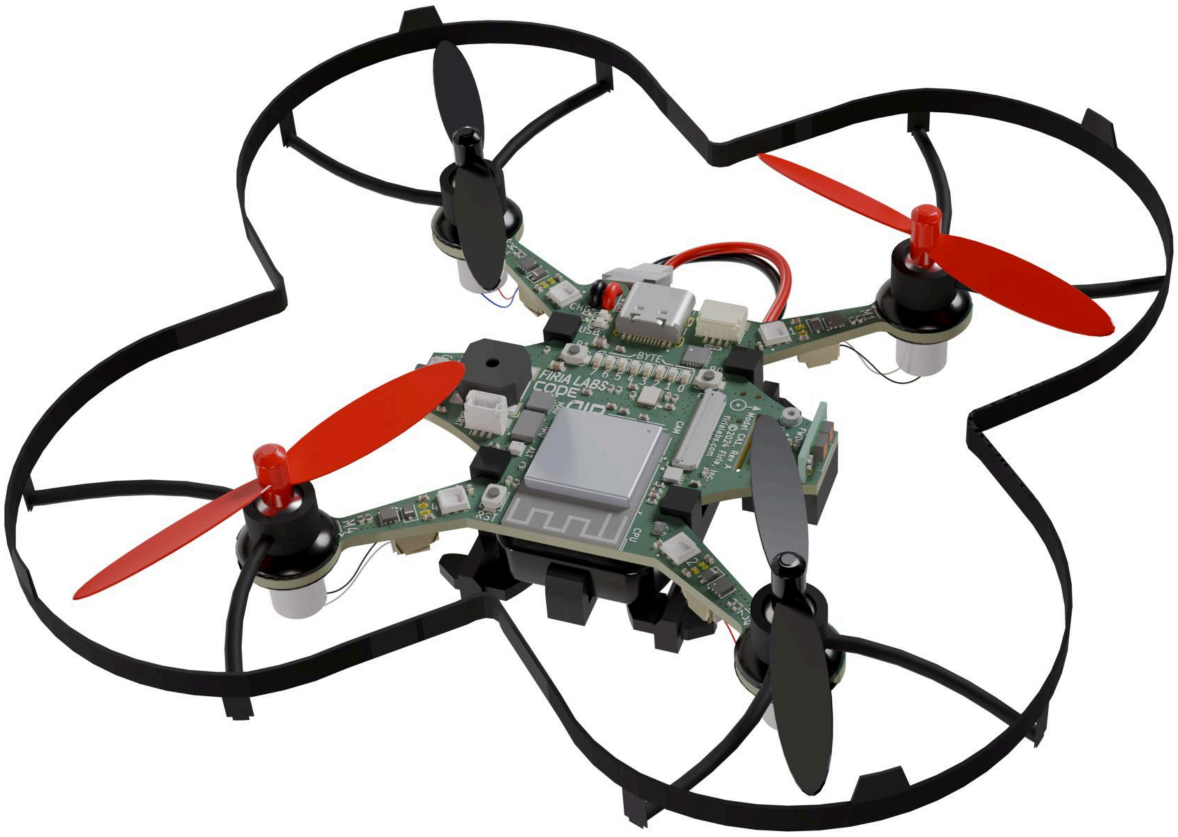
FIRIA LABS

Curriculum Guide



Mission Pack:

Fly with Python

Table of Contents

Fly with Python Overview

This mission pack teaches Python coding by controlling our next-generation autonomous educational drone, designed to integrate AI and Python programming into the classroom. This curriculum will inspire students with engaging STEM projects that blend advanced technology with hands-on learning.

CodeAIR and the Fly with Python Mission Pack revolutionize learning with its onboard neural network and camera. This allows students to deploy machine learning models for real-time object and pattern recognition. CodeAIR operates autonomously, providing a seamless and interactive coding experience without the need for constant radio control.

Pre-Mission Assignment

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 CodeAIR

Students get an introduction to CodeAIR. The device gets connected, and students run some code!

Mission 3: Pre-Flight Check

All systems GO for takeoff? Test out some basic Python coding to control CodeAIR's lights and sounds!

Mission 4: Flight Safety

Good drones don't jump into the sky unexpectedly! Code a safety interlock to be sure your CodeAIR stays put until you are ready to fly.

Mission 5: Hovering Flight

Take off! Get your drone in the air for an autonomous hover, and learn more Python coding as you explore the Laser Range sensors.

Mission 6: Navigate

Navigate the skies by getting a fix on the ground with CodeAIR's flow sensor. Explore the capabilities and limitations of onboard positioning systems.

Mission 7: Attitude Control – coming soon!

Check your attitude, dude! No, not your demeanor…I'm talking about your drone's airframe here!

Suggested Pacing Guide

The way you present the Mission Pack and its missions, objectives and assignments is up to you. Students can work independently and at their own pace to complete the missions. Or you can work together as a class and engage in extensions and cross curricular activities with the missions. A suggested timeline is provided below. It includes time for students to explore and investigate the code. Completing the mission pack can take more or less time, depending on your students, their interests and previous experience, the time you have to spend on it, and many other factors. It is up to you!

This timeline is based on a 5-day, 45-minute class period.

Week 1	Unplugged activities: See CS Unplugged in the learning portal				
Week 2	Mission 1 Mission 1 Review	Mission 2 Mission 2 Review	Mission 2 Extensions or cross curricular	Mission 3 Obj 1-4 & Q2	Mission 3 Obj 5-6
Week 3	Mission 3 Obj 7 and Q3	Mission 3 Obj 8	Mission 3 Extensions or cross curricular	Mission 3 Review Unit 1 Remix Step 1	Unit 1 Remix Step 2-3
Week 4	Unit 1 Remix Step 4	Unit 1 Remix Step 5	Unit 1 Review / Assessment	Mission 4 Obj 1-4	Mission 4 Obj 5-6 / Review
Week 5	Mission 4 Extensions or cross curricular	Mission 5 Obj 1-2	Mission 5 Obj 3	Mission 5 Obj 4-5	Mission 5 Obj 6
Week 6	Mission 5 Obj 7, Q2	Mission 5 Obj 8	Mission 5 Obj 9, Q3	Mission 5 Obj 10	Mission 5 Extensions or cross curricular
Week 7	Mission 5 Review Unit 2 Remix Step 1	Unit 2 Remix Step 2-3	Unit 2 Remix Step 4	Unit 2 Remix Step 5	Unit 2 Review / Assessment
Week 8	Mission 6 Obj 1	Mission 6 Obj 2, Q1	Mission 6 Obj 3	Mission 6 Obj 4, Q2	Mission 6 Obj 5
Week 9	Mission 6 Obj 6, Q3	Mission 6 Obj 7	Mission 6 Extensions or cross curricular	Mission 6 Review Extensions	
Week 10					
Week 11					
Week 12					
Week 13					
Week 14					
Week 15					
Week 16					

Mission 1: Welcome	Time Frame: 20-30 minutes
Mission 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 ● Quiz after Objective 4 ● Add tool to toolbox (Objective 3) ● Review questions	**Success Criteria** ☐ Complete each Objective Goal ☐ Complete Mission 1 Assignment

Standards

CSTA Standards Grades 9-10	CSTA Standards Grades 11-12	AI4K12 Standards Grades 9-12
● 3A-IC-27	● 3B-AP-20	

Student Materials	Teacher Resources
● Laptop/computer with Chrome browser ● Student account / Flying with Python license ● Getting Started with CodeSpace (slides) ● CodeAIR Mission 1 Assignment (PDF)	● License and Dashboard Resources ● CodeAIR Mission 1 Assignment Answers ● CodeAIR Mission 1 Review Questions

Vocabulary

Objective	The steps in the mission; has a goal to accomplish.
Text editor	Where you type the code.
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

Teacher Notes
- Create a class on the teacher dashboard. Generate a join code for the class section to give students.
- This lesson is the first lesson in all the mission packs. If your students have completed other Firia mission packs, 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.
- Review questions can be used as a class review, made into a Kahoot!, or used to create an exam in your learning management system.

Extensions	Cross-Curricular
● none	● none

Mission 2: Introducing CodeAIR	**Time Frame:** 30-40 minutes
Mission Goal: Students will learn about the CodeAIR drone and its LED lights. **Learning Targets** ● I can identify the parts of CodeAIR. ● I can connect CodeAIR to CodeSpace. ● I can create and save a file in CodeSpace. ● I can control LEDs on CodeAIR with code.	**Key Concepts** ● The battery is charging anytime CodeAIR is plugged in. ● CodeAIR must be turned on to connect to CodeSpace and run code. The switch is tiny! ● Code typed into the text editor is automatically saved to your personal file-system. ● CodeTrek is like your own personal tutor, guiding you through programming challenges. ● Python code is case sensitive. Punctuation is important! That is the first thing to check if an error occurs in code.
Assessment Opportunities ● Quiz after Objective 2 ● Quiz after Objective 8 ● Complete the program *Lights1* ● Mission 2 Assignment ● Mission 2 Review questions	**Success Criteria** ☐ Identify key parts of CodeAIR ☐ Connect CodeAIR to CodeSpace ☐ Create a program file with code ☐ *Lights1* works correctly and runs without errors ☐ Complete Mission 2 Assignment

Standards

CSTA Standards Grades 9-10	CSTA Standards Grades 11-12	AI4K12 Standards Grades 9-12
● 3A-CS-01	● 3B-CS-02	

Student Materials	**Teacher Resources**
● Laptop/computer with Chrome browser ● CodeAIR drone ● CodeAIR Mission 2 Assignment (PDF or doc) ● CodeAIR Flying Guide (PDF or doc)	● Getting Started instructions ● CodeAIR Flying Guide (PDF or doc) ● CodeAIR Mission 2 Assignment Answers ● CodeAIR Mission 2 Review Questions

Vocabulary

CodeAIR	A high performance micro-drone that's fully programmable in Python.
LEDs	Light emitting diodes; tiny and efficient electronic components that produce light. CodeAIR has 8 blue indicator LEDs, numbered 0 through 7.
Buttons	Momentary push buttons on CodeAIR that you can program. The user interface push buttons are B0 and B1.
Motors	Brushed DC motors that are electric and power the propellers to lift CodeAIR into the air and maneuver it around.
Static Electricity	A charge that builds and can cause a jolt and spark that happens when grounded.
CPU	Central Processing Unit; the brain of the computer. It interacts with all the peripherals.
Peripheral	Devices that give input or output. CodeAIR's peripherals include LED lights, speaker, motors, sensors and pushbuttons.
Code	Instructions to the computer.

Comment	Code that doesn't get run; notes in the code about what you are doing.
Module	Also known as a library; it contains pre-loaded code, like functions and methods, that can be used once the module is imported.

New Python Code

`from codeair import *`	Import the codeair library; all built-in code specific to CodeAIR

`leds.set(num, brightness)`
`leds.set(0, 50)`
	Sets the user LED at a brightness level.
	In this example, LED 0 is set to 50 percent brightness

Teacher Notes

- The assignment document follows the mission and is a place for guided notes. You can print the document for the students (PDF) or assign it digitally through your LMS (doc).
- If you or your students have trouble running code, follow the CodeAIR Flying Guide. The CodeAIR must be connected and turned on, even if you are not flying the drone.
- Review questions can be used as a class review, made into a Kahoot!, or used to create an exam in your learning management system.
- Extensions and cross-curricular projects are included to enhance the concepts in the mission. You can use the extensions to extend students' programming experience. A remix is not explicitly planned, but you can add one as an option to give students additional learning. A remix is planned after Mission 3.

Extensions

- Use the LEDs to display a pattern by either turning some lights on and keeping some off, or using different levels of brightness.
- Discuss abstraction and how it is used in the hardware of CodeAIR.

Cross-Curricular

- **SCIENCE:** CodeAIR has several LEDs. Research what an LED is and how it works.
- **LANGUAGE ARTS:** Write a technical document that explains the parts of CodeAIR and their functions.

Mission 3: Pre-Flight Check

Time Frame: 60-90 minutes

Mission Goal: Students will learn how to conduct a pre-flight check of CodeAIR.

Learning Targets
- I can control CodeAIR's lighting system.
- I can program the speaker to add sounds.
- I can program the onboard lighting system to show the colors of the international *Aircraft Position Lighting* scheme.

Key Concepts
- To slow down computer code, which runs very quickly, you need a delay. In Python, use a sleep() function to slow down the action.
- You can use editor shortcuts to copy and paste code.
- A loop, like the *while True:* statement or *for* loop, repeats a block of indented code.
- CodeAIR has a speaker that produces beeps in different frequencies and duration.
- CodeAIR has 8 pixel LEDs that can light up any color. They are numbered 0-7.

Assessment Opportunities
- Quiz after Objective 2
- Quiz after Objective 4
- Quiz after Objective 7
- Complete the program *CycleLEDs*
- Complete the program *Melody*
- Complete the program *SkyLights*
- Complete the program *RunningLights*
- Mission 3 Assignment
- Mission 3 Review questions

Success Criteria
- ☐ Use a loop to repeat a block of indented code
- ☐ Use the speaker to play music
- ☐ Control the pixel LEDs
- ☐ *RunningLights* works correctly and runs without errors
- ☐ Complete Mission 3 Assignment

Standards

CSTA Standards Grades 9-10	CSTA Standards Grades 11-12	AI4K12 Standards Grades 9-12
• 3A-CS-01 • 3A-CS-02 • 3A-CS-03 • 3A-DA-11 • 3A-AP-15 • 3A-AP-16 • 3A-AP-21	• 3B-AP-11 • 3B-AP-15 • 3B-AP-22 • 3B-AP-23	

Student Materials
- Laptop/computer with Chrome browser
- CodeAIR drone and USB cable
- CodeAIR Mission 3 Assignment (PDF or doc)
- CodeAIR Flying Guide (PDF or doc)

Teacher Resources
- CodeAIR Mission 3 Assignment Answers
- CodeAIR Mission 3 Review Questions
- CodeAIR Flying Guide (PDF or doc)

Vocabulary

Pre-flight Checks	Going through a detailed checklist before every flight. The list includes lighting systems, safety devices, control surfaces, engines and navigation sensors.
Embedded systems programming	Writing code that goes in a tiny microcontroller embedded in an electronic device.
Sequence	Code that runs one line at a time, in order; sequential.
Sleep	Controlling the pace of code execution by using a delay timing tool.

While loop	A statement that tells Python to repeat a block of code indented beneath it as long as the given condition is true.
Condition	An expression that evaluates to True or False (Boolean).
Infinite loop	Repeat a block of code while a condition is always True – doesn't end.
Constants	Named values that don't change during program execution. Constants are usually defined at the top of program code, just below imports.
Scientific pitch notation	A method of specifying musical pitch by combining a musical note name (A-G) and a number identifying the pitch's octave (0-9).
RGB color	Digital colors made up of (RED, GREEN, BLUE) light. The three colors each have a brightness from 0-255 to create many colors. The values of each color are stored in a list or tuple.
Pixel LEDs	Multi-colored LEDs that can be controlled by the CPU; also known as NeoPixels.
Range	A sequence of numbers you can iterate over. When the range() function is used, the iteration starts at the first number (or default 0) and stops one integer before the last number. seq = range(5) will iterate over 0, 1, 2, 3, and 4.
Iteration (iterating)	Repeating, or iterating, through a sequence of some kind. Examples of a sequence are a range of numbers, a range of colors, a list or a tuple.
For loop	A way to perform iteration.
Standard Navigation Lights	An international standard color scheme to indicate the orientation of the craft. Helpful for anti-collision. The lights are solid (not flashing) and positioned as follows: ● Green for starboard side (right) ● Red for port side (left) ● White for the backend, or tail

New Python Code

`from time import sleep`	Import the time library to access built-in timing functions like sleep
`leds.set(0, 0)`	Turn off an led; use a brightness of 0

`leds.set(0, 50)`
`sleep(1)`
`leds.set(0, 0)`
`sleep(1)`
	Blink an LED for 1 second intervals.

`while True:`	Infinite loop (instruction ends with a colon (:) and block underneath is indented)

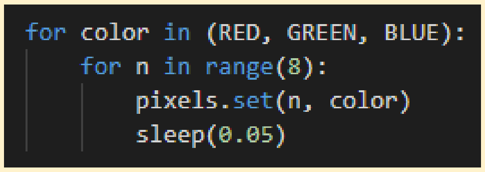
`speaker.beep(frequency, duration)`
`speaker.beep(440, 200)`
	Play a note (or sound) using CodeAIR's speaker
	In this example, the frequency is 400 and the duration is 200 ms

`D5 = 587`	Constant definition
`leds.set_status(50)`	A single LED positioned near the USB connector. The command needs a single argument for brightness.

`COLOR_LIST = (BLACK, BROWN, RED, ORANGE, YELLOW, GREEN, BLUE, PURPLE, GRAY, WHITE, CYAN, MAGENTA, PINK, LIGHT_GRAY, DARK_GREEN, DARK_BLUE)`	Standard color definitions that are included in the **codeair** library from the **colors** module.

`pixels.set(num, color)`
`pixels.set(0, RED)`
	Set a pixel LED to a specific color
	In this example, pixel 0 is set to RED

`pixels.set(0, BLACK)`	Turn off a pixel LED. Here, color names are in ALL CAPS because they are included in the pre-defined COLOR_LIST.
`for n in range(8):`	For loop that starts at 0 and goes up to but not including the ending value. In this example, the iteration would be 0, 1, 2, 3, 4, 5, 6 and 7.

 `for color in (RED, GREEN, BLUE):`
`    for n in range(8):`
`        pixels.set(n, color)`
`        sleep(0.05)`
	Loop for turning pixels red, then green, then blue.

`pixels.set(TOP_FRONT_LEFT, RED)`	Pixels can be designated with a number or constant for position: BOTTOM_FRONT_LEFT, BOTTOM_FRONT_RIGHT, BOTTOM_REAR_LEFT, BOTTOM_REAR_RIGHT, TOP_FRONT_LEFT, TOP_FRONT_RIGHT, TOP_REAR_RIGHT, BOTTOM_REAR_RIGHT
`pixels.fill(WHITE, brightness=50)`	Turns all 8 pixels WHITE at brightness 50. This code is much shorter than turning on all 8 pixels individually.

`sleep(1.0)`
`pixels.fill(WHITE, brightness=50)`
`sleep(0.02)`
	Strobe

Teacher Notes

- The assignment document follows the mission and is a place for guided notes. You can print the document for the students (PDF) or assign it digitally through your LMS (doc).
- If you or your students have trouble running code, follow the CodeAIR Flying Guide. The CodeAIR must be connected and turned on, even if you are not flying the drone.
- Review questions can be used as a class review, made into a Kahoot!, or used to create an exam in your learning management system.
- Extensions and cross-curricular projects are included to enhance the concepts in the mission. You can use the extensions to extend students' programming experience. A remix is not explicitly planned, but you can add one as an option to give students additional learning. A remix is planned after Mission 3.

Extensions

- Use the LEDs and pixel LEDs to display a light show. Add music.
- Discuss abstraction and how it is used in the hardware of CodeAIR.
- Have students do a code review. In the code review, students should select a loop to evaluate and discuss the efficiency, correctness and clarity of the loop.

Cross-Curricular

- **SCIENCE:** Pixel LEDs use red, green and blue lights to make colors. Research how light produces color.
- **MATH:** Use math skills to plan a light show with specific timing: how long to sleep, and how long to display the lights.
- **LANGUAGE ARTS:** Compare and contrast the different types of loops.

Unit 1 Remix Project	Time Frame: 2-3 hours
Remix Project Goal: Students will use the skills and concepts they learned in Missions 1, 2 and 3 to create their own project.	**Remix Project Outline:** Follow the five-steps of the design process to design a remix project (see Remix 1 Project Planning Guide).
Remix 1 Project Assessment Opportunities ● Remix 1 Project Planning Guide ● Peer reviews / Gallery walk ● Remix 1 Project Rubric and/or Checklist ● Submit Remix 1 Program	**Mission 1, 2, and 3 Summative Assessment** ● Unit 1 Review Questions ● Mission 1 Review Questions ● Mission 2 Review Questions ● Mission 3 Review Questions

Supplementary Materials *(available at learn.firialabs.com)*
- Unit 1 Remix Project Slides (slides)
- Unit 1 Remix Project Planning Guide – includes space for 2 peer reviews (PDF and doc)
- Optional: Peer Review Form (PDF and doc)
- CodeAIR Remix Mastery Rubric (PDF and doc)
- CodeAIR Remix Standards Rubric (PDF and doc)

CSTA Standards
The Unit 1 Remix Project covers the standards for Mission 2 and Mission 3. In addition, the remix gives students an opportunity to work collaboratively in a team. These additional standards are met when students work collaboratively in teams and incorporate feedback from users.
- 3A-AP-19: Systematically design and develop programs for broad audiences by incorporating feedback from users.
- 3A-AP-22: Design and develop computational artifacts working in team roles using collaborative tools.
- 3A-AP-23: Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.
- 3B-AP-17: Plan and develop programs for broad audiences using a software life cycle process.
- 3B-AP-20: Use version control systems, integrated development environments (IDEs), and collaborative tools and practices (code documentation) in a group software project.

Teacher Notes	**Unit 1 Remix Project Rubric Checklist**
● The slides can be used to introduce the remix project and give students ideas for their own project. ● The planning guide can be printed for each team (PDF) or given digitally (DOC). ● You can use the checklist or a rubric for assessment. They can all be modified to fit your needs and expectations. ● You can modify the rubric checklist if there are things you don't want to require, or if there are other requirements you want to add. ● Two rubrics are provided. Both may include standards or requirements that are not covered in Unit 1. You can modify the rubrics as needed.	☐ New file is used and filename is descriptive ☐ Define and use a constant ☐ Define and use a variable ☐ Turns on at least one blue LED ☐ Turns on at least one pixel LED ☐ Uses a sleep delay one or more times ☐ Uses a while loop ☐ Uses a for loop ☐ Plays at least two notes using the speaker ☐ Is different from required programs ☐ Includes comments and whitespace for readability ☐ Code runs with no errors

Mission 4: Flight Safety	**Time Frame:** 60-90 minutes
Mission Goal: Students will program a set of safety procedures that can be used in future drone flights. **Learning Targets** ● I can write code to arm the drone for take-off. ● I can write code for a warning indicator. ● I can understand the quadcopter power system.	**Key Concepts** ● Safety procedures can ensure the user is safe before a drone take-off. ● CodeAIR has two input buttons. ● An *if* and *elif* statements allow a branch of indented code to run, based on a condition. ● The *break* statement stops a loop. ● Buttons can accidentally record two presses. Code to "debounce" a button is necessary. ● Create a function for code you want to reuse.
Assessment Opportunities ● Quiz after Objective 2 ● Quiz after Objective 5 ● Complete the program *safety.py* ● Mission 4 Assignment ● Mission 4 Review questions	**Success Criteria** ☐ Use an *if* statement and *break* statement to stop a loop when a button is pressed ☐ Use code to "debounce" button presses ☐ Create a user-defined function ☐ *safety.py* works correctly and runs without errors ☐ Complete Mission 4 Assignment

Standards

CSTA Standards Grades 9-10	CSTA Standards Grades 11-12	AI4K12 Standards Grades 9-12
● 3A-CS-03 ● 3A-DA-11 ● 3A-AP-13 ● 3A-AP-16 ● 3A-AP-17 ● 3A-AP-21	● 3B-NI-04 ● 3B-AP-08 ● 3B-AP-14 ● 3B-AP-15 ● 3B-AP-22 ● 3B-AP-23	

Student Materials	**Teacher Resources**
● Laptop/computer with Chrome browser ● CodeAIR drone and USB cable ● CodeAIR Mission 4 Assignment (PDF or doc) ● CodeAIR Flying Guide (PDF or doc)	● CodeAIR Mission 4 Assignment Answers ● CodeAIR Mission 4 Review Questions ● CodeAIR Flying Guide (PDF or doc)

Vocabulary

Quadcopter Safety Guidelines	Steps to take to ensure personal safety when working with a drone. They include wearing protective gear, avoiding contact with moving parts, and operating in a clear area.
Safety interlock	A safety measure that prevents an electronic device from starting until an event is triggered, like a button press.
UX	User experience; it encompasses the navigation of a product and how easy to use it is.
Branching 'if' statement	A programming control structure that lets code do something different if a certain condition happens, like a button press. This is different from sequential or iteration.
Bounce	When the metal contacts of an electronic input peripheral like a button bounce a few times before coming to a rest. This problem could mean a peripheral is read more than once.

Function	Reusable code with a name. Making reusable components is a major goal of software engineering. Once a button is defined, it must be called before the code is executed.
Variable	A named value used in code, like a box with a label. Use the variable name instead of the value. A value can be any data type, including a number, a string (text) or a Boolean.
Torque	Rotational force produced by motors. When torque is produced, there is a naturally occurring force in the opposite direction.

New Python Code

`buttons.was_pressed(BTN_0)`	Checks to see if B0 was pressed since the last check.
`break`	Breaks out of the nearest enclosing loop

`if buttons.was_pressed(BTN_0):`
`     break`
	If statement (branching) that checks for a button press. buttons.was_pressed(BTN_0) is either True or False.

`while True:`
`    if buttons.was_pressed(BTN_0):`
`        break`
	If statement in an infinite loop. The code waits for a button press before moving to the next line of code.

`pixels.fill(YELLOW)`	Sets all 8 pixels to YELLOW (built-in color)
`pixels.off()`	Turn off all 8 pixels

`sleep(0.1)`
`buttons.was_pressed()`
	Debounce the buttons.
	This line of code resets both buttons!

`from flight import *`	Imports the flight module so you can use built-in functions, like motor_test()

`motor_test(True)`
`motor_test(False)`
	Start / stop a motor test that spins the motors but not fast enough to lift off.

`def button_arm():`	Function definition. The indented block below is the code of the function. A function definition always has () for parameters, even if none are given.
`return do_launch`	Returns (sends) data from the function back to the code that called it. A return ends the function.
`if button_arm():`	Call the function button_arm(), which returns a True or False value
`set_param(‘motorPowerSet.m2’, 30000)`	Set motor (m2) power (30000)
`set_param(‘motorPowerSet.enable’, 1)`	Enable power to the motors
`set_param(‘motorPowerSet.enable’, 0)`	Disable power to the motors

Teacher Notes

- The hints in Objective 1 give suggestions for troubleshooting programming errors, and also for using Chat GPT to understand code. You may want to review the hints with students.
- The assignment document follows the mission and is a place for guided notes. You can print the document for the students (PDF) or assign it digitally through your LMS (doc).

- If you or your students have trouble running code, follow the CodeAIR Flying Guide. The CodeAIR must be connected and turned on, even if you are not flying the drone.
- The buttons to press are very tiny and can be hard to see. They are located just next to the first and last blue LEDs. The blinking lights help identify which button to press in the correct order.
- Review questions can be used as a class review, made into a Kahoot!, or used to create an exam in your learning management system.
- Extensions and cross-curricular projects are included to enhance the concepts in the mission. You can use the extensions to extend students' programming experience. A remix is planned after Mission 5.

Extensions

- Have students use ChatGPT to annotate a section of their code. Then have students do their own annotation of a different section of code.
- Have students do a code review. They should go through the button_arm() function and describe what each section does.

Cross-Curricular

- **SCIENCE:** The mission discusses torque and Newton's 3rd Law. Study these topics in more depth.
- **LANGUAGE ARTS:** After using Chat GPT to annotate a section of code, write a paragraph that describes how artificial intelligence drives many software and physical systems.
- **LANGUAGE ARTS:** Make a list of troubleshooting strategies used to identify and fix errors.

Mission 5: Hovering Flight	**Time Frame:** 90-180 minutes
Mission Goal: Students will program CodeAIR to avoid walls and seek an exit by utilizing its sensors. **Learning Targets** ● I can use the console output print() statement. ● I can control CodeAIR with the MotionCommander interface. ● I can use blocking and non-blocking functions. ● I can measure distances with CodeAIR's laser rangers. ● I can work with variables in Python.	**Key Concepts** ● You can save important functions as a custom module. Then import and use the module in future programs. ● CodeAIR uses a high-level flight control interface called MotionCommander that uses sensors to maintain stable flight. ● Functions can be blocking or non-blocking. ● When it is connected to CodeSpace, CodeAIR can print information to the console. ● A variable is a name you attach to an object so your code can work with it. ● You can use a variable to give your code memory. Update the variable to use it as a counter.
Assessment Opportunities ● Quiz after Objective 5 ● Quiz after Objective 7 ● Quiz after Objective 9 ● Save *safety.py* as a custom module ● Complete the program *Hover* ● Complete the program *Rangers* ● Complete the program *Ceiling* ● Complete the program *Theremin* ● Complete the program *HallMonitor* ● Complete the program *Avoidance* ● Mission 5 Assignment ● Mission 5 Review questions	**Success Criteria** ☐ Create the *safety.py* custom module ☐ Use MotionCommander commands to fly ☐ Read sensors and unpack their values into variables ☐ Use the CodeAIR and variables as a people counter ☐ Use the blue LEDs as counters ☐ *Avoidance* works correctly and runs without errors or bugs ☐ Complete Mission 5 Assignment

Standards

CSTA Standards Grades 9-10	CSTA Standards Grades 11-12	AI4K12 Standards Grades 9-12
● 3A-CS-03 ● 3A-DA-11 ● 3A-AP-13 ● 3A-AP-15 ● 3A-AP-16 ● 3A-AP-17 ● 3A-AP-18 ● 3A-AP-21 ● 3A-IC-24 ● 3A-IC-26	● 3B-DA-06 ● 3B-AP-10 ● 3B-AP-14 ● 3B-AP-15 ● 3B-AP-16 ● 3B-AP-21 ● 3B-AP-22 ● 3B-AP-23	● 1-A-ii

Student Materials	**Teacher Resources**
● Laptop/computer with Chrome browser ● CodeAIR drone and USB cable ● Poster board or similar material to use as a "wall" ● CodeAIR Mission 5 Assignment (PDF or doc) ● CodeAIR Flying Guide (PDF or doc)	● CodeAIR Mission 5 Assignment Answers ● CodeAIR Mission 5 Review Questions ● CodeAIR Flying Guide (PDF or doc)

Vocabulary

Module	An external source of code that is outside your own source file; also known as a library.
Custom module	Some code that is in the same folder as your program and can be accessed by importing it.
docstring	A documentation string; a comment at the top of the file that explains what it does. Use triple quotes (‘ ’ ’ or “ ” ”) to start and stop a docstring.
Console	A window that lets you see output from print() statements.
Blocking function	A function that runs one line at a time, blocking your code from continuing until they are finished. Examples: steady() and sleep()
Non-Blocking function	A function that starts a movement, then returns to the code. Another command must be sent to change or stop the movement.
OODA loop	“Observe, orient, decide, act” – a continuous loop run by the CPU to keep the drone flying at a desired altitude.
Variable	A name attached to an object so your code can work with it. The object can be any data: a number, text, tuple, etc.
Tuple	Ranger data – a set of three values indicated with parenthesis (forward, up, down).
Polling	Repeatedly checking something to see if anything has changed.
Actuator	A device that receives signals and responds with a specific action. When flying a drone, the motors are actuators that receive input from sensors.
Updating a variable	Changing the value of a variable with assignment. An example is to increment a count by 1.
Dead reckoning	A type of navigation that calculates the vehicle’s position based on its known starting point, speed, direction and elapsed time.
Sensor-based navigation	A type of navigation that is adaptive, relying on real-time data gathered by sensors to detect obstacles and adjust course accordingly.
REPL	Repeat evaluate print loop; using the console to interactively enter commands and view outputs in a text format.

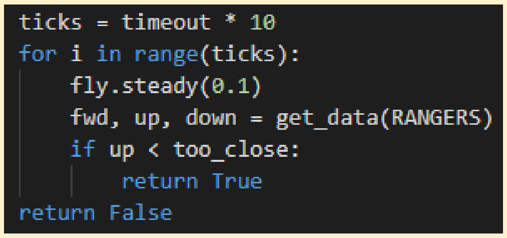
New Python Code

`'''This is a docstring'''`	Document string that should go at the top of any module
`fly.take_off(height_meters)`	Ascend to given height altitude
`fly.steady(seconds)`	Hover, allows code to pause while keeping the flight controller running
`fly.land()`	Descend to the floor
`fly.forward(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`fly.back(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)

`fly.left(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`fly.right(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`fly.up(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`fly.down(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`get_data(RANGERS)`	Returns the (forward, up, down) distance in millimeters

`if up < too_close:`
`   # sound alarm`
	If statement with a condition

`fwd, up, down = get_data(RANGERS)`	Unpack the data from the rangers from the three values in the tuple to three variables


`ticks = timeout * 10`
`for i in range(ticks):`
`    fly.steady(0.1)`
`    fwd, up, down = get_data(RANGERS)`
`    if up < too_close:`
`        return True`
`return False`
	Algorithm for polling with a blocking function.
	In this example, timeout is a parameter that receives seconds from an argument. The polling will happen ten times per second.

`speaker.beep(400, 0)`	Causes the beep to play continuously. Requires speaker.off() to stop the beep.
`count = count + 1`	Incrementing or updating a variable
`leds.set_mask(0, 0)`	Turn off all the blue LEDs
`fly.start_forward()`	Non-blocking function that starts moving forward at the default velocity and returns immediately so the next instruction can be executed
`fly.stop()`	Stop any motion and hover
`fly.turn_left(degrees)`	A blocking function that turns the drone degrees left
`if count == 8:`	Checks if **count** is the same as 8. If it is, a branch of code is executed.

Teacher Notes

- This mission is lengthy and includes several programs and concepts. Take your time, even an entire week. Using several days for the Objectives will give your students plenty of time to explore and try things.
- In Objective #1, students add code to their custom module. The __*name*__ and '__*main*__' use two underscores _ _. You may need to mention this to students to avoid errors.
- The floor space and lighting really do make a difference in how well CodeAIR can track its movements. If you have a lot of drift, move to different flooring or add a pattern to the floor to help with navigation.
- In Objective #6, think safety first! Use a file folder or clipboard as an obstacle above the drone instead of a hand. You don't want fingers in the blades.
- The CodeTrek in Objective #8 shows a step-by-step algorithm for counting people. This mission uses several algorithms. This is a good time to have a discussion about algorithms and practice writing them.
- Extensions and cross-curricular projects are included to enhance the concepts in the mission. You can use the extensions to extend students' programming experience. A remix is planned after Mission 5.

Extensions

- Create a more elaborate security system after Objective 5 and/or Objective 6.
- Code a different ending to fix the bug after Objective 10.
- Have students do a code review. Pick any of the six programs to review.
- The programs in this mission use algorithms to solve a problem. Discuss what an algorithm is. Practice writing algorithms.

Cross-Curricular

- **MATH:** In Objective #3 the drone flies forward a given distance. Create a chart of distances entered in code and actual distance flown. Calculate the difference. Write an equation to predict future flights.
- **SCIENCE:** The theremin was invented during research into proximity sensors. Study the physics involved in the musical instrument. Try building and playing your own theremin.
- **LANGUAGE ARTS:** Write a paragraph that explains when to use a loop and when to use an if statement in code.
- **LANGUAGE ARTS**: Evaluate the ways computing impacts personal or cultural practices.
- **LANGUAGE ARTS:** Make a list of troubleshooting strategies used to identify and fix errors.

Unit 2 Remix Project	Time Frame: 2-3 hours
Remix Project Goal: Students will use the skills and concepts they learned in Missions 4 and 5 to create their own project.	**Remix Project Outline:** Follow the five-steps of the design process to design a remix project (see Remix 2 Project Planning Guide).
Remix 2 Project Assessment Opportunities ● Remix 2 Project Planning Guide ● Peer reviews / Gallery walk ● Remix 2 Project Rubric and/or Checklist ● Submit Remix 2 Program	**Unit 2 Summative Assessment** ● Unit 2 Review Questions ● Mission 4 Review Questions ● Mission 5 Review Questions
Supplementary Materials *(available at learn.firialabs.com)* ● Unit 2 Remix Project Slides (slides) ● Unit 2 Remix Project Planning Guide – includes space for 2 peer reviews (PDF and doc) ● Optional: Peer Review Form (PDF and doc) ● CodeAIR Remix Mastery Rubric (PDF and doc) ● CodeAIR Remix Standards Rubric (PDF and doc)	
CSTA Standards The Unit 2 Remix Project covers the standards for Mission 4 and Mission 5. In addition, the remix gives students an opportunity to work collaboratively in a team. These additional standards are met when students work collaboratively in teams and incorporate feedback from users. ● 3A-AP-19: Systematically design and develop programs for broad audiences by incorporating feedback from users. ● 3A-AP-22: Design and develop computational artifacts working in team roles using collaborative tools. ● 3A-AP-23: Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs. ● 3B-AP-17: Plan and develop programs for broad audiences using a software life cycle process. ● 3B-AP-20: Use version control systems, integrated development environments (IDEs), and collaborative tools and practices (code documentation) in a group software project.	
Teacher Notes ● The slides can be used to introduce the remix project and give students ideas for their own project. ● The planning guide can be printed for each team (PDF) or given digitally (DOC). ● You can use the checklist or a rubric for assessment. They can all be modified to fit your needs and expectations. ● You can modify the rubric checklist if there are things you don't want to require, or if there are other requirements you want to add. ● Two rubrics are provided. Both may include standards or requirements that are not covered in Unit 1. You can modify the rubrics as needed.	**Unit 2 Remix Project Rubric Checklist** ☐ New file is used and filename is descriptive ☐ Use at least one variable ☐ Turns on at least one blue LED ☐ Turns on at least one pixel LED ☐ Uses at least one loop ☐ Uses at least one if statement ☐ Moves the drone using flying statements ☐ Uses data from at least one sensor ☐ Has a purpose and is different from required programs ☐ Includes comments and whitespace for readability ☐ Code runs with no errors

Mission 6: Navigate	**Time Frame:** 120-180 minutes
Mission Goal: Students will program CodeAIR to fly autonomously using sensor data. **Learning Targets** ● I can explore positioning systems with the flow sensor for x, y tracking. ● I can observe and analyze flow sensor accuracy by flying CodeAIR in a square. ● I can conduct a battery check to ensure safe and sustained flight. ● I can customize selectable operations to control code behavior during runtime. ● I can experiment with flight parameters including height, distance and velocity.	**Key Concepts** ● The flow sensor is an optical device that can discern patterns on the ground and report movement in two directions. ● The sensor tracks motion by summing up the changes in x and y over time. ● You can use an f-string to determine how information is printed. ● Data from multiple sensors is needed for accurate flight control. ● CodeAIR's battery is charging whenever it is plugged in. ● The 8 LEDs can use binary patterns to display integers between 0 and 255. ● Handle exceptions in code using a *try* block.
Assessment Opportunities ● Quiz after Objective 2 ● Quiz after Objective 4 ● Quiz after Objective 6 ● Complete the program *FlowTracker* ● Complete the program *SquareUp* ● Complete the program *SquareTurns* ● Complete the program *BattTest* ● Complete the program *utility.py* ● Complete the program *RouteSelect* ● Mission 6 Assignment ● Mission 6 Review questions	**Success Criteria** ☐ Use flow sensor data to track position ☐ Fly CodeAIR in a square ☐ Perform a battery check ☐ Create a user interface using binary patterns ☐ Handle exceptions using a *try* block ☐ *RouteSelect* works correctly and runs without errors or bugs ☐ Complete Mission 6 Assignment

Standards

CSTA Standards Grades 9-10	CSTA Standards Grades 11-12	AI4K12 Standards Grades 9-12
● 3A-CS-01 ● 3A-CS-03 ● 3A-DA-09 ● 3A-AP-13 ● 3A-AP-14 ● 3A-AP-16 ● 3A-AP-17 ● 3A-AP-18 ● 3A-AP-21 ● 3A-IC-26	● 3B-CS-02 ● 3B-DA-06 ● 3B-DA-07 ● 3B-AP-10 ● 3B-AP-14 ● 3B-AP-15 ● 3B-AP-16 ● 3B-AP-21 ● 3B-AP-22 ● 3B-AP-23	● 1-A-ii ● 1-B-i ● 1-C-ii

Student Materials	**Teacher Resources**
● Laptop/computer with Chrome browser ● CodeAIR drone and USB cable ● Metric tap measure or ruler ● CodeAIR Mission 6 Assignment (PDF or doc) ● Optional: Mission 6 Flight Data (spreadsheet) ● CodeAIR Flying Guide (PDF or doc)	● CodeAIR Mission 6 Assignment Answers ● CodeAIR Mission 6 Review Questions ● CodeAIR Flying Guide (PDF or doc)

Vocabulary

Positioning systems	A system for determining the position of an object in space.
Flow sensor	A sensor used to track horizontal movement across a surface; essential for stable hover and precise navigation.
Delta	A change in position, symbol from the Greek letter Δ and used in math and science to represent change.
String	A data type that is a sequence of characters all strung together. Can be numbers, letters, spaces, whatever!
Format string	A template for printing a string using replacement fields that are designated with {curly braces}. This allows actual arguments to be inserted into the template.
MotionCommander Interface	An interface between Python code and the flight controller that provides a high-level flight control interface and uses onboard sensors to maintain stability.
Sensor fusion	When data from multiple sensors is combined. For example, combining altitude data from the laser ranger with the data from the flow sensor.
Under load	When a battery is powering peripherals, like CodeAIR's motors, it is under load.
Binary numbers	How computers deal with digits. Two states (on and off) are represented with 1 and 0.
Bit	A single binary digit (1 or 0)
Byte	An 8-bit number
Selectable operations UI	A user interface that uses one button to scroll through a menu and another button to confirm the current selection, and then start the action.
Exceptions	Errors that might happen during your program execution.
External positioning system	A positioning system that uses something outside the drone to determine location, such as GPS or a fixed-location beacon.

New Python Code

`dx, dy = get_data(FLOW)`	Read data from the flow sensor; returns the change in x direction and change in y direction
`print(x, y)`	A simple print statement that converts data to strings and displays them on the console
`print("Flow Sensor Output")`	Print a string text on the console
`print(f"x={x}, y={y}")`	F-string with replacement fields in curly braces
`abs(x)`	Returns the absolute value of x
`vbatt = power.battery_voltage(10)`	Read battery voltage, average 10 samples
`amps = power.charger_current()`	Read charging current when connected with USB
`usb_connected = power.is_usb()`	Returns True if currently powered by USB

`value = 0b1001`	Set the value to 9 using binary
`leds.set_mask(255, 50)`	Set BYTE LEDs to 255 (on) with brightness = 50
`leds.set_mask(0b10101010, 50)`	Set BYTE LEDs using binary
`If __name__ == '__main__':`	Detects when this program is being run as the “main program” instead of an import
`try:`	A block of code that executes when no error occurs, or until an error occurs.
`except:`	A block of code that lets your program respond to an error without crashing.

Teacher Notes

- The last objective requires students to create a chart and log the results of several test flights. The assignment document has a chart they can use. Alternatively, students can use a spreadsheet (one is provided) and use the spreadsheet for more extensive data analysis.
- This mission involves several programs, and each one can be used for experimentation. Don’t rush, and give students plenty of time to try their code with different values and in different environments. You could easily spend a class period on almost every single objective.
- Objective #4 has some important instructions after the goal is met. Students need to add their battery check to *safety.py* and then run the code to reinstall it on CodeAIR. Don’t let students skip this part!
- The test flights during the Objectives will need plenty of unobstructed room. Clear out as much furniture as you can, or use hallways. Also check the flooring. Use flooring with a pattern for best results.
- Extensions and cross-curricular projects are included to enhance the concepts in the mission. You can use the extensions to extend students’ programming experience. A remix is planned after Mission 7.

Extensions

- Fly the drone in a different shape: triangle, hexagon, etc.
- Create a more elaborate battery check that shows the charging amp when plugged in or the battery charge when disconnected.
- Have students do a code review. Pick any of the six programs to review.
- Write algorithms for the functions, or explain what each line of code does. Discuss parameters and arguments.

Cross-Curricular

- **MATH:** The code uses built-in math functions like abs() and min(). Discuss other built-in math functions that could be used in code. Practice using them in practical problems.
- **MATH:** Objective #5 introduces binary numbers. Practice converting binary and decimal numbers. Explore other number systems besides decimal and binary.
- **MATH:** Several objectives involve testing the drone’s flight capabilities. The last objective uses a chart to document the tests. Do an in-depth analysis of the data. Create charts for other objectives and analyze the data.
- **LANGUAGE ARTS:** Make a list of troubleshooting strategies used to identify and fix errors.

Appendix A: Required Resources

Computer Resources

Each student will need:

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

Software Resources

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

Physical Resources

The missions can be completed by individual students or student pairs utilizing pair programming. Each student or student pair will still need a CodeAIR and license for the curriculum.

- CodeAIR comes with one USB to USB-C connecting cable.
- Materials needed but not included:
    - Poster board or something to simulate a "wall" to enclose the CodeAIR
    - A meter stick or tape measure for determining distance in meters

Notes

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

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 Fly with Python 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 is provided, but it offers just a suggestion of the time you can spend on each mission and objective. We encourage you to take your time with each objective and not rush. Give students time to explore and investigate; it is all learning!

Remixing and 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. This gives 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. Remixes and extensions are also an excellent opportunity for students to synthesize their learning and create their own projects. We highly recommend including extensions and/or remixes into your pacing calendar.

Some extension activities include code reviews and class discussions. These are also excellent opportunities to extend the learning of your students. Cross-curricular activities are also suggested, that can play on student interests and help connect the world of computer science and programming to the real world.

Managing a Class

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

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

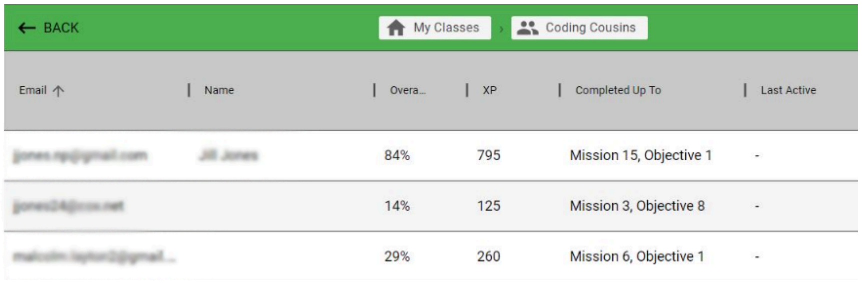
If you are a **Google Classroom** teacher, you can import assignments from CodeSpace into your classes. For instructions, go to "Virtual Tools with CodeSpace" in the Teacher Resources for Python with Robots.

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

Here are the basics of the CodeSpace Teacher Dashboard

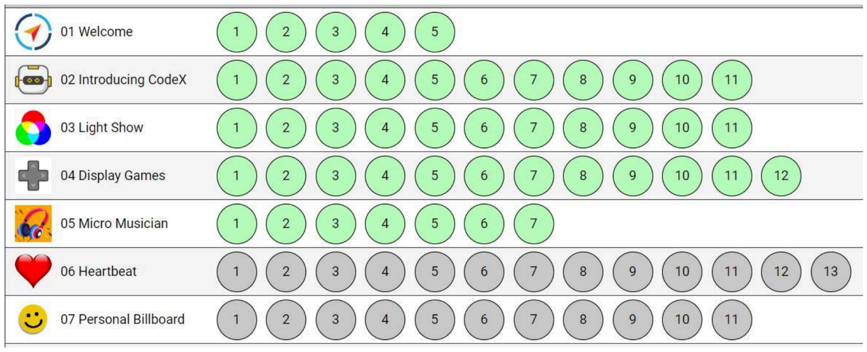
- Log in to CodeSpace and from HELP, select CLASS DASHBOARD
- Once you are in the dashboard, click + in the green bar, top right corner, to add a class.
- Assign each class a name, and allow members to join with a join code.
- You can assign Google Classroom as your LMS.
- After the class is created,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

Class dashboard



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

Formative Assessment

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, submitting programs from intermediate objectives, exit tickets, and assignments.

Summative Assessment

Each lesson has at least one embedded quiz, a list of vocabulary and a list of Python coding. A multiple choice test, short answer test, or programming test can be created with the information. Also, the final coded mission can be submitted as an assessment. Each mission also has suggestions for extensions. An extension can be required for assessment. Two rubrics are available to use with any mission, remix or final project. They are the mastery rubric and the standards-based rubric, available at learn.firialabs.com.

Program Extensions

If you require an extension or remix, you can use one of the two provided rubrics or the checklist. Rubrics, the checklist, and any additional assessment device can be used as a PDF form to fill out, or made into a digital form. Either rubric can be modified to meet the specific goals of the mission or your own requirements. Make a copy and edit as needed. You can also customize the rubric by adding specific requirements or assigning point values before students begin.

Student-Teacher Conferencing

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

Peer Feedback

Before students submit a remix project or extension, they should complete a peer review. This may take modeling a few times before students do it correctly. Remind students that revising is just as important here as it is in English class. These revisions can lead to great conversations during the conferencing process. They should go through the rubric and test the program just as you would. This will give them the chance to find and correct mistakes before doing a student-teacher conference. Each remix planning guide includes two peer reviews. A peer review form is included at learn.firialabs.com.

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.