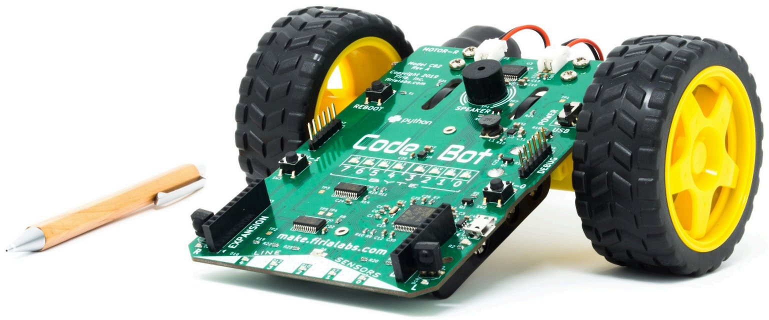
Python with Robots

Middle School Curriculum Guide



Unit 4

Underneath the Hood

Unit 4: Underneath the Hood (8-10 hours)

Students learn about CodeBot's proximity sensors. Sensor data is used to control CodeBot, tracking and chasing an object. Then students learn about internal sensors for checking battery voltage and CPU temperature. They also learn about the accelerometer, what it is and how to use it for detecting movement. Their skill with functions is improved and expanded, including global and local variables.

Summary of Mission 7: This mission is divided into three lessons, each approximately one class period. During the mission students learn about proximity sensors and how to access the data. The final program is an autonomous robot that can auto-calibrate and then track and follow an object.

> **Lesson 1:** Proximity sensors are introduced. Students learn about power and threshold settings and use pre-defined functions to determine the best settings.
>
> **Lesson 2:** Students write two functions that work together to auto-calibrate the proximity sensors.
>
> **Lesson 3:** Students add movement to the program by controlling the motors with the proximity sensor data. The lesson deviates from CodeSpace for Obj 11 and gives an alternative that discusses abstraction and has students define and call two more functions for the program.

NOTE: This unit skips Mission 8 Navigation. It is all about wheel encoders. The code in the objectives uses a lot of functions, lists, and complex math, which can be difficult for middle school or beginner students. The recommendation is to skip the mission entirely, or return to the mission after Unit 4 as a bonus.

Summary of Mission 9: This mission is divided into three lessons, each approximately one class period. Students learn about internal system sensors: battery health, temperature and accelerometer.

> **Lesson 1:** Students learn about checking battery health. They write a program that displays how much power is left in batteries.
>
> **Lesson 2:** Students learn about system temperature. They write a program that can sound an alarm if the temperature is too hot or too cold.
>
> **Lesson 3:** Students learn about the accelerometer: what it is and how to read it. They write two programs. The first orients CodeBot to always point its nose up. The second is a guard bot that detects motion.

Unit 4 Remix: The remix project can be an extension of a current program or a completely new project. Students can work individually with a partner, or in teams.

Unit 4 Classroom Materials that are provided with each lesson:

- **Lesson Plan:** A detailed lesson plan is provided for each lesson. It includes learning targets, success criteria, vocabulary and new code. It also has teaching tips for each objective.
- **Mission Slidedeck:** Each lesson includes PowerPoint slides for teacher-led instructions. You can use them to guide students through the material. At times the slides introduce things a little differently, and often give more examples. The code can be slightly different than CodeTrek. All goals will be met.
- **Mission Log:** Each lesson has an assignment, called a mission log, for students to complete as they go through the lesson. It includes warm-up and wrap-up questions as well as questions along the way for guided notes. An answer key for each mission log is provided.
- **Code Solutions:** Sample code for each final project, as well as some of the objectives, can be

found in the learning portal with the other teacher materials.

- **Kahoot! Review:** Each lesson has a Kahoot! that reviews the concepts and codes.

Unit 4 Preparation:

- Have a computer / laptop with the Chrome web browser for each student.
- Make sure students can log into CodeSpace at http://make.firialabs.com with their email address.
- Have a CodeBot and USB cable for each student or programming pair.
- Ruler for measuring distance of an object from CodeBot (can use the “White Paper with Ruler” in the learning portal).
- Four AA batteries are needed for each CodeBot.
- Objects for CodeBot to detect (folders, toys, small and large objects)
- A sturdy surface for CodeBot to sit on, inclined.

Assessment:

Mission 7 Obj 1-3 Kahoot Review	Mission 7 Obj 4-7 Kahoot Review	Mission 7 Obj 8-11 Kahoot Review
Mission 9 Obj 1-4 Kahoot Review	Mission 9 Obj 5-7 Kahoot Review	Mission 9 Obj 8-12 Kahoot Review

Coming soon: Unit 4 reviews and multiple choice exams are provided.

Unit 4 Vocab Kahoot Review	Unit 4 Coding Kahoot Review	Unit 4 Vocab Test (MS Form)	Unit 4 Coding Test (MS Form)

Standards addressed in this unit:

CSTA Standards Grades 6-8	CSTA Standards Grades 9-10	CSTA Standards Grades 11-12
• 2-CS-02 • 2-CS-03 • 2-DA-07 • 2-DA-08 • 2-DA-09 • 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-13 • 2-AP-14 • 2-AP-15 • 2-AP-16 • 2-AP-17 • 2-AP-19	• 3A-CS-01 • 3A-CS-03 • 3A-DA-11 • 3A-AP-13 • 3A-AP-14 • 3A-AP-15 • 3A-AP-16 • 3A-AP-17 • 3A-AP-18 • 3A-AP-19 • 3A-AP-21 • 3A-AP-23	• 3B-CS-02 • 3B-DA-06 • 3B-AP-10 • 3B-AP-12 • 3B-AP-14 • 3B-AP-16 • 3B-AP-17 • 3B-AP-21 • 3B-AP-22 • 3B-AP-23

Basic Lesson Plans for each lesson included here.

For complete lesson plans, see each mission in the Learning Portal.

MISSION 7: Hot Pursuit Lesson 1 (Objectives 1-3)	Time Frame: 45-50 minutes
Project Goal: Students use proximity sensors to detect an object. **Learning Targets** ● I can use proximity sensors to detect objects. ● I can indicate a detected object using the proximity sensor LEDs. ● I can experiment with light and dark surfaces to find the best power and detection threshold settings for each surface. ● I can use a pre-defined function to scan multiple settings to find the best detection threshold for a given power setting.	**Key Concepts** ● CodeBot uses the Infrared Proximity Sensor system to detect objects in its path. ● A detection threshold of 0-100% controls how much light is needed for a True detection. If you decrease the thresh value, the 'bot works well, even on a white surface. ● An emitter power level setting from 1 to 8 (high power) controls the brightness of CodeBot's IR "flashlight." ● The prox.detect(power, thresh) function lets you adapt to different environments. ● The prox.range(num_scans, power) function scans multiple settings to find the best threshold for the given power.
Assessment Opportunities ● Mission 7 Lesson 1 Log ● Submit completed program ***HotPursuit*** ● Mission 7 Obj. 1-3 Review Kahoot!	**Success Criteria** ☐ Use the basic function prox.detect() to detect objects in front of the sensors. ☐ Use the reading from prox.detect() to turn on/off the proximity sensor LEDs. ☐ Measure the detection distance for a white surface using different power and thresh values. ☐ Measure the detection distance for a black surface using different power and thresh values. ☐ Use the function prox.range() to find the best threshold for detecting IR light on a white surface. ☐ Use the function prox.range() to find the best threshold for detecting IR light on a black surface.
Teacher Materials in Learning Portal ● Mission 7 Lesson 1 Slides ● Mission 7 Lesson 1 Log ● Mission 7 Lesson 1 Answer Key	**Additional Resources** ● Mission 7 Obj. 1-3 Review Kahoot! ● HotPursuit_obj3 sample code (learning portal) ● HotPursuit_ext1 sample code (learning portal)

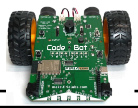
Vocabulary
- **Proximity sensors:** Infrared (IR) sensors that can detect nearby objects based on the reflected IR light.
- **Detection threshold:** How much light is needed for a True detection. For proximity sensors, the range is 0%--100%.
- **Emitter power level:** The brightness of CodeBot's IR flashlight, with settings from 1 (low) to 8 (high power).

New Python Code

`p = prox.detect()`	Calling this function pulses the emitter and detects reflected IR light. It returns a tuple of two bools: (left, right).

`p = prox.detect(power, thresh)`
`leds.prox(p)`
	Turn on the LED below each sensor if an object is detected. Example: if p = (True, False), the left LED is turned on and the right LED stays off.

`p = prox.detect(power, thresh)`
`leds.prox(p)`
	Proximity detection with optional parameters (power, threshold). The power parameter is a range from 1 to 8 for the IR brightness. The



	threshold is the detection sensitivity, with a range from 0%-100%.
`sensed = prox.range(10, power)`	The function scans multiple sensitivity levels to find the lowest detection threshold (0%-100%). It has four optional parameters: num_samples, power, range_low, range_high. We use the first two. 10 is the highest num_sample possible.

Real World Applications
Detecting objects is used by many electronic objects you might use every day, without even thinking about it. Some examples are given below. Can you think of even more?

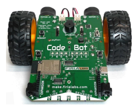
- touchless faucets
- soap dispensers and hand dryers
- automatic doors
- vehicle navigation and safety systems
- factory automation systems

Teacher Notes:

- The code for Objectives 1 and 2 is the same as CodeTrek. You can use the slide or CodeTrek.
- The code for Objective 3 is different from CodeTrek. You can type the code directly in the console panel without adding to the code. However, the last goal in the objective will not be met until you add a line of code. This is detailed in the slides.
- Students will need a white surface and a black or very dark surface. Regular printer paper and black construction paper work well.
- For Obj. 2, students need to measure distance. You can put a ruler next to the surfaces, or use the paper PDF in the teacher resources, which has a metric ruler. Just put the black construction paper over the white paper for the second table.
- This lesson can run long if students are doing a lot of testing. You can condense the lesson and have students do less testing, stretch the lesson a little bit into the next day, or use the extension for Objective 3 instead of typing all the code in the console panel.

Extensions / Cross-Curricular:

- Modify the while True loop to cycle through the power range and print the results to the console. This is a fast way to test all the powers for many different surfaces and can be an alternative to using the console panel exclusively in objective 3.
- **SCIENCE:** Learn more about IR light or proximity sensors. Research where proximity sensors are used.
- **MATH:** Create a chart from the data collected during objective 2 or objective 3. Use a different color for the different surfaces.
- Supports **language arts** through reading instructions, guided notes, and reflection writing.



MISSION 7: Hot Pursuit
Lesson 2 (Objectives 4-7)

Time Frame: 45-50 minutes

Project Goal: Students can write calibration functions so the ‘bot can adapt to its environment.

Learning Targets
- I can write code that calibrates the detection sensitivity threshold.
- I can write a function for the calibration.
- I can write a function that calibrates the power level.

Key Concepts
- Using auto calibration functions for **power** and **thresh** allows the ‘bot to adapt to a new environment.
- An accepted programming convention is to group functions together, typically at the beginning of a program.
- Print statements can take multiple arguments. It converts them to strings and prints them back-to-back to the console. Each argument is separated with a comma.

Assessment Opportunities
- Mission 7 Lesson 2 Log
- Submit completed program ***HotPursuit***
- Mission 7 Obj. 4-7 Review Kahoot!

Success Criteria
- ☐ Use an if statement to detect a button press
- ☐ Use if statements to determine the best detection sensitivity threshold
- ☐ Define a function to auto-calibrate thresh
- ☐ Define a function to auto-calibrate power
- ☐ Use the global command when assigning values to global variables inside a function
- ☐ Use multiple arguments in a print statement
- ☐ Test the program with multiple surfaces and record the results

Teacher Materials in Learning Portal
- Mission 7 Lesson 2 Slides
- Mission 7 Lesson 2 Log
- Mission 7 Lesson 2 Answer Key

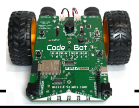
Additional Resources
- Mission 7 Obj. 4-7 Review Kahoot!
- HotPursuit_obj4 sample code (learning portal)
- HotPursuit_obj6 sample code (learning portal)
- HotPursuit_obj7 sample code (learning portal)
- HotPursuit_ext2 sample code (learning portal)

Vocabulary
- **Calibrate:** Using sensor readings to determine values of variables; adapting code to the environment using data.
- **Automate:** Use technology, like a computer, to do a task automatically.
- **Default:** A pre-selected option.
- **Globals:** Variables defined outside of a function; they are available during the entire program and can be accessed throughout the entire program.
- **Locals:** Variables defined inside a function; they only exist while the function is running and can only be accessed in the function.

New Python Code

`if sensed[LEFT] > 0:`
`    det = sensed[LEFT]`
	After reading the proximity sensors (sensed), check only the LEFT sensor reading to see if it detected a reflection. If so, assign its sensitivity value to a variable.

`det = min(det, sensed[RIGHT])`	The math function min() finds the lowest, or minimum, value of the arguments. In this example, it selects the lowest value of either the current det or the sensitivity value of the RIGHT sensor.



`if det > 100:`
`    thresh = 100`
`else:`
`    thresh = det - 5`
	Bug fix to make sure thresh is the correct value.

`global thresh`
`global power`
	Keeps a global variable as global, even when it is assigned a value inside a function.

`while power < 9:`
`    cal_thresh()`
`    if thresh < 100:`
`        break`
`    power = power + 1`
	Loop that cycles through the range of powers to determine the best power level for the proximity sensors.

`print("Power=", power)`	Print statement with multiple arguments.

Real World Applications

Many technical devices use auto-calibration to ensure the device maintains accuracy. Here are some examples:

- Adaptive cruise control on cars auto-calibrate to measure distances accurately and prevent ghost braking.
- Smartphone accelerometers and gyroscopes automatically calibrate for orientation.
- In labs, automated pipettes can perform self-calibration to ensure the volume of liquid matches the display.

Teacher Notes:

- This lesson is all about auto-calibration, which was also introduced in Mission 6. If you skipped auto calibration (Mission 6 Objective 8), then all the information you need is included here. If your students completed Obj. 8, then this will be review.
- This lesson can run long if students are doing a lot of testing, or if calibration is a new topic. You can consider running the lesson over two class periods and including a cross-curricular activity or the extension as well.
- This lesson goes over global and local variables. If your students did Mission 6 Obj. 8, then it will be review. Otherwise, you may want to take time on this and make sure students understand the concept.
- The code used during the lesson is slightly different from CodeTrek. All goals will be met.
- Students will need different surfaces for testing. You can use the Testing Surfaces paper with black/white/gray options, or any color or surface.
- There is a quiz after Objective 5. It doesn't really review the concepts from 4 and 5. You have the option of skipping the quiz, or going over it together as a class.

Extensions / Cross-Curricular:

- After Obj 7, the code can still throw an error if the 'bot is not under a surface. Fix the bug by using an if statement.
- **SCIENCE/MATH:** Experiment with the proximity sensor using different lighting, from very bright to pretty dark. See what range of IR light and sensitivity work best in different conditions. Make a chart of the power/thresh.
- **MATH:** The proximity sensors can give a different reading, even with the same surfaces and lighting. Run the auto-calibration sequence several times without moving the 'bot and record all the power/thresh values. Find the mean, median and mode for the surfaces.
- Supports **language arts** through reading instructions, guided notes, and reflection writing.

MISSION 7: Hot Pursuit
Lesson 3 (Objectives 8-11)

Time Frame: 45-50 minutes

Project Goal: Students will use proximity sensors to program the 'bot to track and chase an object.

Learning Targets
- I can apply previous knowledge of the motors to rotate and face an object moving in front of CodeBot.
- I can follow an algorithm to track an object.
- I can apply a safety feature by toggling a Boolean variable.
- I can follow an algorithm to chase an object.
- I can apply abstraction to the program by defining and calling functions.

Key Concepts
- The previous concepts of controlling CodeBot's motors can be applied and used with proximity sensors. This includes enabling the motors and supplying power to the LEFT and RIGHT motors.
- Both prox.detect() and prox.range() return a tuple. The first returns a tuple of bools and the second a tuple of integers. You can access the LEFT sensor value with p[LEFT] and the right value with p[RIGHT].
- The logical operator "not" takes only one argument. It can be used to toggle a Boolean variable and useful for turning something on/off.
- Functions are a form of procedural abstraction, a fundamental concept used in programming and STEM.

Assessment Opportunities
- Mission 7 Lesson 3 Log
- Submit completed program ***HotPursuit***
- Mission 7 Obj. 8-11 Review Kahoot!

Success Criteria
☐ Access the LEFT and RIGHT proximity sensor detection reading

☐ Use motor controls in an if statement to track an object

☐ Toggle the motors on and off using a Boolean variable and the logical operator "not"

☐ Modify the if statement to chase an object

☐ Define a function for driving the 'bot

☐ Define a function for toggling the motors

Teacher Materials in Learning Portal
- Mission 7 Lesson 3 Slides
- Mission 7 Lesson 3 Mission Log
- Mission 7 Lesson 3 Mission Log Answer Key

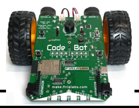
Additional Resources
- Mission 7 Obj. 8-11 Review Kahoot!
- HotPursuit_obj10 sample code (learning portal)
- HotPursuit_obj11 sample code (learning portal)
- HotPursuit_ext3 sample code (learning portal)
- HotPursuit_ext4 sample code (learning portal)

Vocabulary
- **Toggle:** Flip from True to False, or False to True; on to off or off to on.
- **Not:** A logical operator that is used to toggle a Boolean variable.
- **Abstraction:** The process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics.

New Python Code

`if p[LEFT] and p[RIGHT]:`
`    motors.run(LEFT, 40)`
`    motors.run(RIGHT, 40)`
	Control the motors if both proximity sensors detect an object.

`elif p[LEFT]:`
`    motors.run(LEFT, 0)`
`    motors.run(RIGHT, 20)`
	Turn the 'bot either left or right if only the LEFT or RIGHT proximity sensor detects an object.



`elif p[RIGHT]:`
`    motors.run(LEFT, 20)`
`    motors.run(RIGHT, 0)`	

`go_motors = False`	Define a Boolean variable that will toggle the motors enabled on/off.

`if buttons.was_pressed(0):`
`    go_motors = not go_motors`
	Use the “not” logical operator to toggle the value of the Boolean variable. It will change from True to False or False to True.

`motors.enable(go_motors)`
`leds.user_num(go_motors)`
	Use a Boolean toggle variable to turn on/off the motors and turn on/off a user LED.

Real World Applications

Review the real world applications from Lesson 1. By the end of this mission you can see more clearly how proximity sensors, auto-calibration and movement are used inside many electronic objects you might use every day. Brainstorm even more devices, or discuss how they use these systems.

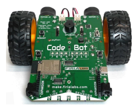
- touchless faucets, soap dispensers and hand dryers
- automatic doors
- vehicle navigation and safety systems
- factory automation systems

Teacher Notes:

- In CodeSpace, objective 11 goes into quite a bit of math and can be complicated. I haven’t found it to be very useful in improving the overall program. The slides introduce an alternative to objective 11 by discussing abstraction and having students write two more functions from their current program. This is the only time when the coding in the slides will not meet the goals of the objective. But it is the last objective, so after students finish you can just unlock the Mission and students can continue with the mission pack.
- Although movement is introduced in objective 8, the ‘bot doesn’t move forward until objective 10. Students can test Obj. 8 and 9 on their desks or on the floor.
- In previous programs, a button press was used to “wait” as a safety feature. Obj. 8 starts without a safety feature. It is added during objective 9..
- The slide deck version of objective 11 introduces the concept of abstraction. A short video from Code.org is used to give an example. You might find your own video or examples of abstraction as an alternative.

Extensions / Cross-Curricular:

- After Obj 7, the code can still throw an error if the ‘bot is not under a surface. Fix the bug by using an if statement.
- Add code to debounce button presses.
- Define a variable for the speed, and use the variable when calling the drive() function. You can use a math operation with the variable for turning.
- **STEM:** Have a lesson on abstraction. Let students come up with and present their own examples of abstraction.
- **ENGLISH LANGUAGE ARTS:** Have students write a story about an electronic device that uses proximity sensors and moves. It can be about how useful and helpful it is, or about how things might go wrong.
- **ENGLISH LANGUAGE ARTS:** Have students write an opinion piece about technology and its effects on society.
- Supports **language arts** through reading instructions, guided notes, and reflection writing.



MISSION 9: All Systems Go! **Lesson 1 (Objectives 1-4)**	**Time Frame: 45-50 minutes**
Project Goal: Students will use system sensors to monitor battery power and display battery health with a UI. **Learning Targets** • I can use system sensors to read battery voltage and power source. • I can print the capacity percentage using a table. • I can calculate the capacity percentage using the equation of a line. • I can create a user interface to display battery health using the power LED.	**Key Concepts** • The 'bot can measure its own battery voltage, and also if the 'bot is powered by USB or batteries. • The data returned by the system sensors can be used to show alerts to avoid problems. Even though CodeBot doesn't have a screen, its LEDs can be used to display sensor data. • A graph of battery discharge can be approximated with a straight line. • Another LED on CodeBot is just above the power switch. It can be turned on/off with code.
Assessment Opportunities • Mission 9 Lesson 1 Log • Submit completed program ***BatteryTest*** • Mission 9 Obj. 1-4 Review Kahoot!	**Success Criteria** ☐ Read battery voltage using system sensors ☐ Indicate battery or USB power using system sensors and the power LED ☐ Display capacity percent using a table ☐ Calculate and display capacity percent using the equation of a line ☐ Use the power LED as a UI to communicate battery health
Teacher Materials in Learning Portal • Mission 9 Lesson 1 Slides • Mission 9 Lesson 1 Mission Log • Mission 9 Lesson 1 Mission Log Answer Key	**Additional Resources** • Mission 9 Obj.1-4 Review Kahoot! • BatteryTest_obj2 sample code (learning portal) • BatteryTest_obj3 sample code (learning portal) • BatteryTest_final sample code (learning portal) • BatteryTest_ext sample code (learning portal)

Vocabulary
- **System sensors:** Sensors that read internal settings, like power and temperature.
- **Under load:** Batteries are being used to power something, like turning on LEDs or running motors.
- **Float (review):** A decimal number, either positive or negative.
- **y = mx + b:** The equation of a straight line.
- **User Interface (UI):** A way for a person and a machine to communicate, which includes screen, keyboards and LEDs.

New Python Code

`system.pwr_volts()`	Returns a float (decimal number) for the current voltage, either from USB or batteries.
`system.pwr_is_usb()`	Returns an integer 0 if the power switch is set to batteries and 1 if USB.
`leds.user(15)`	Turn on the first four user LEDs so the battery is under load.
`leds.user(0)`	Turn off all user LEDs.
`pct = (v / 2) - 2`	Use the equation of a line to calculate the percentage, given the volts.
`leds.pwr(True)`	Turn on the LED indicator for power.

`leds.pwr(False)`	Turn off the LED indicator for power.

Real World Applications
You already use this kind of code daily! Many electronic devices check for battery power to ensure device reliability, extend battery life, and prevent unexpected failures. Some examples are:

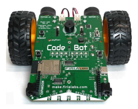
- Cell phones and laptops
- Electric vehicles
- Industrial robots
- Hospital and emergency services equipment

Teacher Notes:

- This lesson follows the instructions in CodeSpace fairly closely. It is chunked into smaller bits of information and it occasionally gives extra examples or definitions.
- The code in this lesson is similar to CodeTrek. It is simplified a little for ease of typing, and some non-required code is left out. All goals will be met.
- The activities include a lot of testing, which uses the console panel for printing results. The console panel can be resized so show more text. Just drag up the gray line above the tab headings for a longer window.
- All CodeBots should have batteries in their battery pack for testing. The batteries don't need to be fresh. In fact, having batteries at a variety of power levels can be more interesting.
- The lesson uses the equation of a line to calculate the percentage. If your students haven't learned the equation of a line yet, you can just give them the code, or have a short lesson on the concept.

Extensions / Cross-Curricular:

- Create an additional user interface to display battery health. Turn on all user LEDs for full power, and decrease the number of turned on LEDs as the battery power decreases. When all user LEDs are off, the battery is close to dead.
- **SCIENCE:** Have a lesson on electricity, voltage and/or battery power. Compare battery power to USB power from the computer.
- **MATH:** This lesson uses percentages. Have a lesson on how to calculate a percent.
- **MATH:** This lesson uses the equation of a line to calculate percentages. Have a lesson on finding the equation of a line. Or have students graph line equations.
- **STEM:** Suppose CodeBot had a display screen. Design a UI that would display battery health and any other system information you are interested in.
- Supports **language arts** through reading instructions, guided notes, and reflection writing.



MISSION 9: All Systems Go! Lesson 2 (Objectives 5-7)

Time Frame: 45-50 minutes

Project Goal: Students will use system sensors to monitor system temperature and display unacceptable temperatures with a UI.

Learning Targets

- I can use system sensors to read system temperature in either Celsius or Fahrenheit.
- I can find the average of several temperatures.
- I can create a UI that uses LEDs to show an alert if the average temperature is too high or too low.

Key Concepts

- The ‘bot can measure its own system temperature.
- Just like the system data for power, the temperature data returned by the system sensors can be used to show alerts to avoid problems. The CodeBot’s LEDs can be used to show alerts.
- The system sensors can measure temperature in either Celsius or Fahrenheit.
- The CPU in the CodeBot CB2 (older model) is more exposed than the CB3 and can more easily be manipulated. At the end of the slide deck, specific instructions are given to test the code with the CB2 that doesn’t work well with the CB3. If you have a newer ‘bot, just skip those two slides.

Assessment Opportunities

- Mission 9 Lesson 2 Mission Log
- Submit completed program ***TemperatureCheck***
- Mission 9 Obj. 5-7 Review Kahoot!

Success Criteria

- ☐ Read system temperature using system sensors
- ☐ Define an empty list for temperature reading
- ☐ Get a sum of all temperatures in the list
- ☐ Return the average of temperatures in a list
- ☐ Define constants for baseline and deadband
- ☐ Turn on/off LEDs as an alert when the temperature is too high or too low

Teacher Materials in Learning Portal

- Mission 9 Lesson 2 Slides
- Mission 9 Lesson 2 Mission Log
- Mission 9 Lesson 2 Mission Log Answer Key

Additional Resources

- Mission 9 Obj. 5-7 Review Kahoot!
- TemperatureCheck_obj5 sample code
- TemperatureCheck_obj6 sample code
- TemperatureCheck_obj7 sample code
- TemperatureCheck_final sample code

Vocabulary

- **Milliseconds:** A millisecond is a thousandth of a second.
- **Ambient:** Surroundings.
- **Append:** Adding a new item to the end of a list.
- **Traverse:** Accessing each item in a list in order.
- **Baseline data:** Starting point used for comparison; original data.
- **Deadband:** In a control system, the range or band of input values where the output doesn’t change.

New Python Code

`bot_temp = system.temp_C()`	Measure temperature in Celsius.
`bot_temp = system.temp_F()`	Measure temperature in Fahrenheit.
`sleep_ms(200)`	Delay program execution for 200 milliseconds.
`samples = []`	Initialize an empty list.
`samples.append(bot_temp)`	Append (add) a new item to a list.

`while i < count:`
`    sum = sum + nlist[i]`
`    i = i + 1`
	Traverse a list using a loop to sum all the items.

`return sum / count`	Return the average without assigning the value to a variable.
`samples.clear()`	Clear a list of all items.

`BASELINE = 25.5`
`DEADBAND = 3.0`
	Define a constant.

`for i in range(5):`	A quick way to loop a block of code five times.

`if t > BASELINE + DEADBAND:`
`    leds.user(0b11111111)`
	Compare temperature for a value that is too high, above the baseline + deadband.

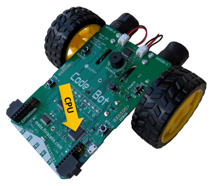
`if t < BASELINE - DEADBAND:`
`    leds.ls(0b11111)`
	Compare temperature for a value that is too low, below the baseline - deadband.

Real World Applications
You already use this kind of code daily! Many electronic devices use a temperature-control system, or thermostat to avoid overheating or overcooling. Some examples are:

- Smart thermostats
- Central heating and air conditioners
- Portable heating and/or cooling units
- Ovens, refrigerators, coffee makers, etc.
- Water heaters, vehicles, medical equipment
- And much more!

Teacher Notes:

- This lesson follows the instructions in CodeSpace fairly closely. It is chunked into smaller bits of information and simplified for clarity.
- The code in this lesson is similar to CodeTrek. It is simplified a little for ease of typing, and it is organized in a way similar to former missions. All goals will be met.
- Three extensions are given for the lesson. The lesson may not take an entire class period. I recommend doing the first extension together. Detailed instructions are given for the extension.
- The older CodeBot model (CB2) is different from the new model (CB3). Its CPU is more accessible. At the end of the lesson, additional instructions are given for testing if you have the CB2 model. Otherwise, skip the slides.



Extensions / Cross-Curricular:

- Fix the check_baseline() function to turn off LEDs when the condition is no longer met.
- Add beeps to the alert system to indicate too high or too low temperatures.
- Add another function that reads the system temperature in Fahrenheit, and use button presses to determine which reading is executed.
- **SCIENCE:** Have a lesson on temperature, what affects temperature, and how temperature affects devices.
- **MATH:** This lesson uses averages. Review the formula for calculating an average and practice with your own data.
- Supports **language arts** through reading instructions, guided notes, and reflection writing.

MISSION 9: All Systems Go! **Lesson 3 (Objectives 8-12)**	**Time Frame: 45-50 minutes**
Project Goal: Students will use input sensors to monitor physical orientation. **Learning Targets** ● I can use system sensors to monitor physical orientation. ● I can use physical orientation to detect motion.	**Key Concepts** ● The accelerometer detects orientation in three dimensions. The CodeBot can be programmed to act on conditions based on its orientation. ● The accelerometer can also be used to detect motion in the CodeBot. Any slight movement can be detected and used for an alarm.
Assessment Opportunities ● Mission 9 Lesson 3 Mission Log ● Submit completed program ***AccelTest*** ● Submit completed program ***GuardBot*** ● Mission 9 Obj. 8-12 Review Kahoot!	**Success Criteria** ☐ Print the 3-axis values from the accelerometer ☐ Write basic code that keeps CodeBot pointed up ☐ Improve the basic code for better control using proportional rotation speed ☐ Detect motion on the X-axis ☐ Detect motion on all 3-axes
Teacher Materials in Learning Portal ● Mission 9 Lesson 3 Slides ● Mission 9 Lesson 3 Mission Log ● Mission 9 Lesson 3 Mission Log Answer Key	**Additional Resources** ● Mission 9 Obj. 8-12 Review Kahoot! ● AccelTest_obj9 sample code (in learning portal) ● AccelTest_obj10 sample code (in learning portal) ● GuardBot sample code (in learning portal)

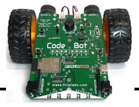
Vocabulary
- **Orientation:** The relative position of something.
- **Accelerometer:** A tiny chip that measures the force of acceleration in 3 directions: x, y and z.
- **Navigation:** Knowing where you are and planning and following a route for where you want to be.
- **Oscillate:** Move or swing back and forth at a steady speed.
- **Proportional:** Change at the same rate.
- **Incline:** Sloping upward.

New Python Code

`accel.dump_axes`	Prints the 3-axis values to the console
`x, y, z = accel.read()`	Reads the current axis values and returns a tuple of integers, ranging from -32767 to +32768
`now = accel.read()`	Read the accelerometer and assign all three values to a tuple.
`now[0]`	Access the X value of the accelerometer reading.
`before = now`	Assign the same tuple (now) to a new variable (before).
`dx = now[0] - before[0]`	Calculate the difference between current reading and previous reading.

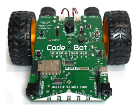
`if abs(dx) > SENS:`
`    alarm()`
	If the difference between readings is more than the sensitivity, sound an alarm.

Real World Applications
Many electronic devices use an accelerometer for orientation and navigation. Some examples are:
- Cell phones



- Smart watches
- Game controllers
- Vehicles
- And much more!

Teacher Notes:	**Extensions / Cross-Curricular:**
• This lesson follows the instructions in CodeSpace fairly closely. It is chunked into smaller bits of information and simplified for clarity. • The code in this lesson is similar to CodeTrek. It is simplified a little for ease of typing, and it is organized in a way similar to former missions. All goals will be met. • Objectives 9 and 10 require an inclined surface. It can be anything that is sturdy and supports a CodeBot. A yearbook, shoebox, cardboard box, anything handy will work. A textbook is great for making the incline. I recommend a small surface that can be picked up and tilted for Obj. 10.	• Fall Detector. Use the accelerometer to detect if the 'bot is falling. • Bump Bot. Move the 'bot forward until it detects an impact. Then rotate a random amount and go again. • **SCIENCE:** Have a lesson on the accelerometer. What are its parts, and how does it work? Research devices that use an accelerometer. • **SCIENCE:** This lesson uses gravity. Have a lesson on gravity, its measurement, etc. • **MATH:** This lesson finds the difference between two readings. The variable dx is used for delta x. Learn about Δ (delta) and how it is used in math. • Supports **language arts** through reading instructions, guided notes, and reflection writing.



Unit 4 Remix Project	Time Frame: 2-5 hours
Project Goal: Students will use the skills and concepts they learned in the unit to create their own project. **Learning Targets** ● I can summarize the programming concepts from Missions 7 & 9. ● I can plan an original program. ● I can create an original program using concepts and code from previous programs. ● I can get feedback on my project.	**Key Concepts** ● Code segments from previous programs can be reused and repurposed in a new project. ● The program development in the planning guide follows the software design process. ● Creating a new project from the beginning, without CodeTrek or starter code, is an excellent way for students to master their learning and gives them an opportunity to express themselves and work on something that interests them.
Assessment Opportunities ● Unit 4 Remix Planning Guide ● Unit 4 Remix Project	**Success Criteria** ☐ Plan an original program ☐ Create an original program ☐ Incorporate feedback in a program
Teacher Materials in Learning Portal ● Unit 4 Remix Project slides ● Unit 4 Remix Planning Guide	**Additional Resources** ● Students can use their previous programs as a guide throughout this project.
Teacher Notes: ● A remix for this unit is optional. However, it is an excellent opportunity for students to create their own original program by doing something that interests them. And the remix project gives students a chance to practice and apply what they have learned. ● Students can work with a partner for this project. Collaboration is an important skill. ● A set of slides is prepared to explain the project and give step by step guidance. The slides also give some suggestions for the project. The suggestions are meant to help students think of their own ideas and should not be required. They can be used for students who are drawing a complete blank, or as inspiration. ● A planning guide is provided to help students know where to start, and to guide them throughout the process. You can modify the planning guide as needed by changing or adding to the questions. ● Consider how you want to end the remix project. You can have students present them to the class, have a "gallery walk" of projects, have students create a slide show about the project, etc. ● A checklist for the remix project is below. You can modify the checklist for specific requirements.	

Remix Project Checklist:

- ☐ Filename is descriptive
- ☐ Uses one or more variables, each with a descriptive name
- ☐ Uses sensor data (proximity, battery, temperature, accelerometer)
- ☐ Controls the CodeBot (movement, turn on LEDs, use speaker)
- ☐ Uses at least one button for input
- ☐ Defines and calls at least one function
- ☐ Includes comments and blank lines for readability
- ☐ Code runs without errors