CS109 Lab 2: Etch-a-Sketch

Fall 2024

Overview

Learning Objectives

In this lab, you'll learn how to use buttons and dials (potentiometers) to interact with your programs. You'll also practice breaking down problems into smaller steps, writing functions, using loops, and working with Boolean values.

By the end of this lab, you should be able to:

1. Read a button press.
2. Read the position of a dial (potentiometer).
3. Send information between your Pico and your laptop.
4. Use your Pico to control a program on your laptop.

Prerequisites

This lab will span multiple class periods. Before starting this lab, you should:

1. Have VSCode installed on your laptop
2. Read through the lab and familiarize yourself with what you will be doing
3. Before you start, accept and clone the Lab 2 assignment from GitHub classroom on a computer that you bring with you to class.
4. Run the same setup command for your project as we did in the last lab: If you are on a Mac, hit command-shift-p. If you are on Windows or Linux, hit ctrl-shift-p. In the text input box that pops up, type MicroPico: Configure MicroPico Project into that textbook and hit Enter. You should see the MicroPico controls show up in the bottom bar.

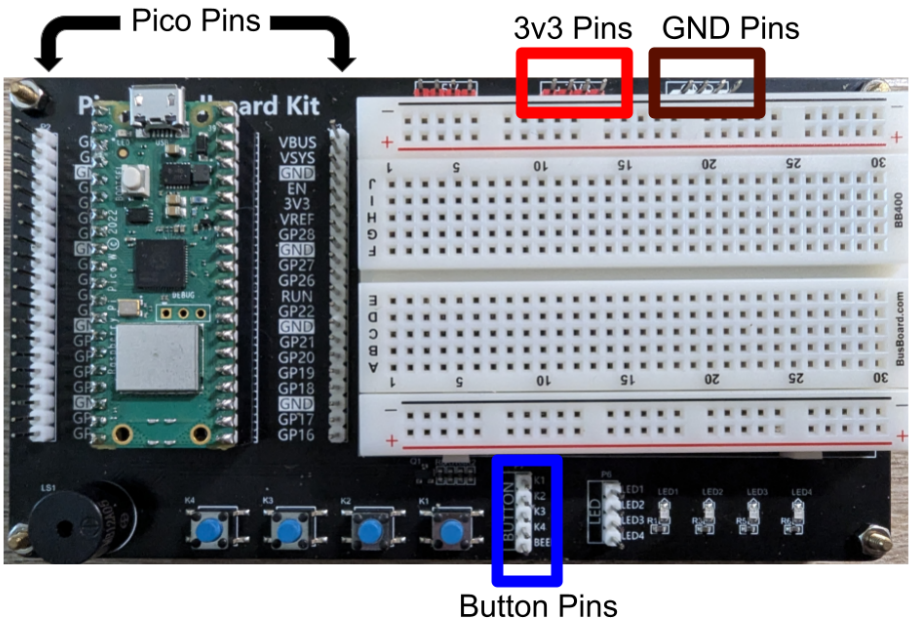


Figure 1: The Pico kit layout

Exploration 1: Reading a Button

In this step, you'll learn how to use a button as an input device on the Pico.

Setup

There are four push-buttons on the bottom of the Pico kit. If you look closely, they are labeled K1-K4. Directly to the right of the buttons are a column of pins labeled Button, with the top pin labeled K1 and the bottom pin labeled BEEP (see Figure 3 (if BEEP is currently not connected to anything, please connect it to one of the GND pins at the top left). Connect one end of a wire to Pin GP15 on the Pico and the other end to the pin labeled K4. This connects the leftmost button to pin number 15 on your Pico.

Reading the Button State

Open the file `button.py`.

1. At the top of this file, import the `Button` class from the `picozero` module (remember, we can do this using `from picozero import Button`).

2. Create an instance of the `Button` class and assign it to a variable called `button1`. When we call the constructor, we'll pass it the number 15 to indicate that the button is connected to pin 15: `button1 = Button(15)`.

3. The `Button` class has an instance variable called `is_pressed`. This is a a `bool` that indicates if the button is currently being pressed or not. Write a print statement that prints out value of `button1.is_pressed`.

4. Run the program on the Pico without pressing the button. What does it print? Now run the program again, but this time hold down the button before running it. Now, what does it print?

5. It's pretty inconvenient to run a program to read the state of a button only once. Let's modify your program so that it continuously prints out whether the button was pressed or not. To do this, we'll use an infinite loop! Put your `print` statement in a while loop where the loop condition is simply `True`.

6. Import `sleep` from the `time` module and call it in the body of your while loop to add a sleep of 1 second between button reads (remember, the sleep function takes a number of seconds to sleep as an input parameter).

7. Run your program again. You should now see a stream of print statements, one per second. Press the button a few times to see the print statements change. Play around with reducing the length of the sleep (try 0.1 seconds) to increase the responsiveness of the button.

Counting Button Presses

Suppose we want to count button presses. Notice that, with a sleep time that gives us a reasonably responsive button reading, the button may register as pressed multiple times during a single button press. How can we detect each individual time the button has been pressed?

To do this, we'll use the following logic: inside your `while True` loop, check to see if the button has been pressed. If it has, then enter another loop where you do nothing until the button has no longer been pressed. This essentially waits until the button has been released to count the button press.

1. Add an accumulator outside your while True loop to count the number of button presses.

2. Add a conditional statement inside your `while True` loop that evaluates to `True` if the button is currently pressed.

3. Inside the conditional, add a while loop that sleeps for 0.1 seconds repeatedly as long as the button is still pressed. NOTE: yes, we are *nesting* a while loop inside a conditional statement which is inside another while loop! The program execution will continue the outer while loop after it finishes the inner one.

4. *AFTER* the inner while loop terminates, add one to the accumulator you are using to count button presses.

5. Change the `print` statement to print out the number of button presses.

Let's use the button to turn the Pico's LED on and off. You might want to refer back to your Lab 0 code if it's helpful.

1. Import the `PicoLED` class from the picozero module. You can import both `Button` and `PicoLED` in one line: `from picozero import Button, LED`

2. Before the `while True` loop, construct a `PicoLED` object and assign it to a variable called `led`.

3. Create a boolean flag variable called `ledState` and initialize it to `False`. Note that this is a boolean accumulator!

4. Right after updating count, add code to do the following: if `ledState` is currently False, then turn the led on *and* set `ledState` to True. Otherwise, turn the led off and set `ledState` to False. Remember, the `LED` class has methods called `on` and `off` that you can use to turn the Pico's led on and off.

5. Run your code on the Pico. Now, when you press the button the built-in LED light should turn on and off.

6. Write a function called `checkButton(button)` that takes a `Button` object, checks if it's pressed, waits until it's unpressed, and returns True if the button was pressed and False if it was not. Hint: this should mostly consist of taking the code you've written and putting it in a function. Now, you can call `checkButton` to detect button presses in the future.

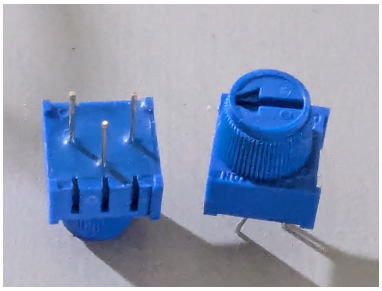


Figure 2: Top and Bottom views of the potentiometer (pot). Mount the pot in the breadboard with the middle pin on the bottom (closest to you), and the two outer pins away from you.

Exploration 2: Reading a Potentiometer

The button is a digital input–it is either pressed or not pressed. Now, let's learn how to use an analog input–one that can take on different values. The input we will use is a rotary knob called a potentiometer. The potentiometers are the blue plastic components with a rotating dial on the top and three pins on the bottom (see Figure 2). The potentiometer position is an int between 0 and 65,535.

Setup

The white plastic board with lots of holes next to your Pico is called a breadboard. It helps us easily connect wires and components without soldering. Each hole in the breadboard is part of a row or column of connected holes.

- With the four buttons on the bottom, the rows are labeled A to E on the near side and F to J on the far side.
- Within each labeled section (A-E or F-J), holes in the same numbered column (like 1, 5, 10, etc.) are connected to each other underneath. This means if you plug a wire into hole A1, it's electrically connected to holes B1, C1, D1, and E1.
- However, rows A-E and F-J are not connected to each other directly. The horizontal gap down the center of the breadboard keeps them separate, allowing you to use one side for one purpose and the other for another.

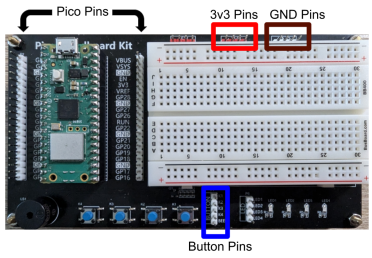


Figure 3: The Pico kit layout (again)

Above the breadboard, you'll see a row of pins labeled 3V3 and another row labeled GND:

- 3V3 provides power to your components. It's like the "+" side of a battery.
- GND stands for "ground" and is like the "-" side of a battery. Hooking a component up to both 3V3 and ground powers it.

Now, let's hook up the potentiometers (dials). Each potentiometer has three metal legs, called "pins":

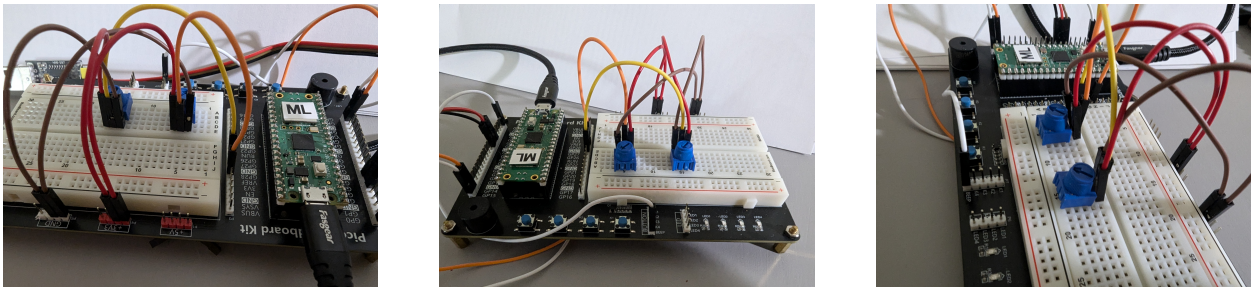
- The two outer pins are for power.
- The middle pin gives a signal based on the dial's position.

Plug the two potentiometers side-by-side into the breadboard so that:

- The middle pin goes into row A (closest to you).
- The two outer pins go into row B (the row above it).
- Leave a little space between the two pots (see Figure 4).

Notice that these three pins span three columns of pins.

- Use a red wire to connect one of the 3V3 pins (for power) above the breadboard to a hole in the same column as the right pin of each potentiometer. This sends power to the potentiometer.
- Then, use a brown wire to connect one of the GND pins (for ground) above the breadboard to a hole in the same column as the left pin of each potentiometer.
- Finally, connect any colored wire between the Pico's GP28 pin (labeled on the right side of the Pico) and a hole in the same column as the middle pin of the left potentiometer. Use another wire to connect the Pico's GP27 pin to a hole in the same column as the middle pin of the right potentiometer.



(a) Back View (b) Front View (c) Side View

Figure 4: Connecting the potentiometers

Reading the Potentiometer

Once you've set up the potentiometers, reading them is pretty straightforward!

1. Open the file `pot.py`.
2. At the top of the file, import the `Pot` class from the `picozero` module. Also import `sleep` from `time`.
3. call the `Pot` constructor to create two different `Pot` objects. Call these `left` and `right` to correspond to the pins you have the left and right potentiometers hooked up to:

```
left = Pot(28)
right = Pot(27)
```

4. The `Pot` class has a method `read_position` that takes no arguments and returns the current position of the potentiometer. For example, `left.read_position()` will return the current position of the left potentiometer.

5. Write a `while True` loop in which you read the positions of each of the two potentiometers, print out both on one line, and sleep for a period of time.

6. Run your program on the Pico. Have one person turn the knobs on each potentiometer and see how the values being printed change. You should see each one vary from close to 0 to close to 65,535, as you turn the dial from all the way to the left to all the way to the right.

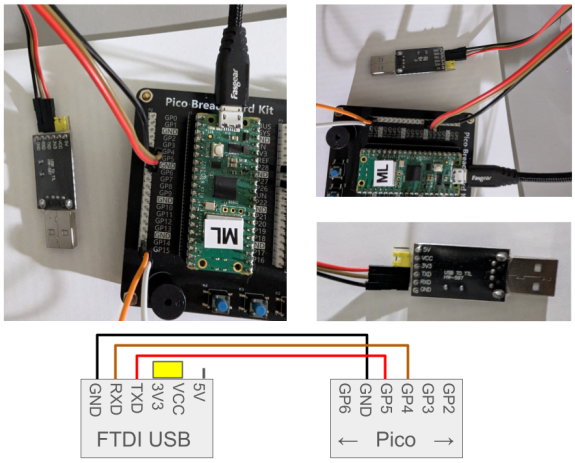


Figure 5: The secondary USB connection dongle

Exploration 3: Communicate with Your Laptop

In the last exploration, you'll learn how to send messages from your Pico to your laptop. To do this, we'll use the USB-to serial dongle in your kit (see Figure). We will call this the secondary USB connection.

Setup

We will use three wires to connect the secondary USB dongle to the Pico. Refer to Figure 5. There should be a yellow box over two of the pins on the dongle. Set the dongle down so that yellow box is on the right and the pin labels are facing up (GND, RXD, TXD, etc). Connect a black wire between the leftmost pin on the dongle, labeled GND, to one of the pins labeled GND on the Pico; connect the second pin, labeled RXD, with a wire (brown in the photo, but yours may differ), to pin GP4, and connect the third pin, labeled TXD, with a wire (red in the photo), to pin GP5.

Don't plug the secondary USB dongle into your laptop yet, we will plug it in later.

We will need an external module called `pyserial` to support this communication. On the laptop which you are connecting to the Pico, in your laptop's terminal **(NOT the Pico terminal)**, run `pip install pyserial` if you are on Windows or `pip3 install pyserial` if you are on Mac or Linux.

Talking to the Pico

We will write and run two programs: one will run on the Pico itself. It will read the two pot values and send them to to your laptop. On your laptop, you'll run a second program that listens to the readings and prints them out.

To do this, we will use two classes: on the Pico, we'll use a class called `Publisher` that's included in `picozero`. On your laptop, we'll use a class called `Receiver` that's defined in `receiver.py`, which is a module included in your starter code.

1. Open the file `publish.py`.
2. Import both `Pot` and `Publisher` from `picozero`. You can do this like so: `from picozero import Publisher, Pot`. Also import `sleep` from the `time` module.
3. Copy your code from `pot.py` to setup and read from your two potentiometers.
4. Create a `Publisher` object using the `Publisher` constructor, and assign it to a variable called `pub`. The `Publisher` constructor takes no arguments (`pub = Publisher()`).

5. The `Publisher` class has a method called `send`. This method, like print, can take a variable number of arguments, however these arguments can only be `ints`. This is convenient, since both of our potentiometer readings are `ints`. For example, if we have two `ints` `x=1` and `y=2`, calling `pub.send(x,y)` would send a message to your laptop containing the `ints` 1 and 2.

6. Inside your `while True` loop, call `pub`'s `send` method, passing it your two potentiometer readings, to send these to your laptop.

7. Put a 0.2 second sleep after sending the message inside your `while True` loop.

8. Now, we'll write a program on the laptop side to receive and print the messages. Open the file `receive.py`.

9. Import the class `Receiver` from the `receiver` module.

10. The `Receiver` constructor takes one argument: a string representing the **name of the USB port** that your secondary USB dongle is attached to. This name will vary from computer to computer. Remember, we haven't yet attached the secondary dongle. In this step, we will figure out what it's name is on your computer.

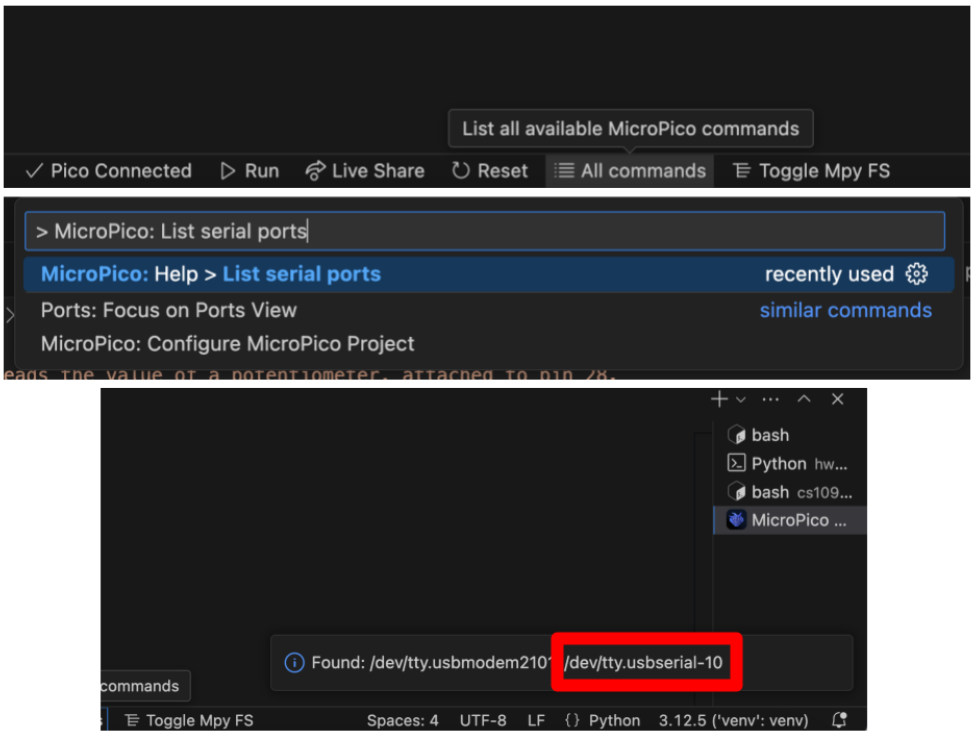


Figure 6: Steps to find the name of the secondary USB port.

To figure out what this is on your computer (See Figure 6):

(a) Either click "All commands" in the Pico control bar at the bottom or type "cmd-shift-p" (Mac) or "ctrl-shift-p" (Windows/Linux).

(b) In the dialog that pops open at the top, type `MicroPico: List serial ports` and click on the first option.

(c) This will pop up a list of the USB port names currently detected in the bottom-right. Port names should look some like `COM4` if you're on Windows or start with `/dev/tty` if you're on Mac. At this point you should only see one name! This is your Pico! Write down the name.

(d) Now plug in your secondary USB dongle. Run the command `MicroPico: List serial ports` again. This time you should see two names! Write down the new name–this is the name of your secondary USB dongle's port. Careful, this is case-sensitive.

11. Now, construct a `Receiver` object by calling the `Receiver` constructer and passing it the name of the USB port as a string. For example, something like: `rec = Receiver("/dev/tty.usbserial-10")`.

12. The receiver class has a method called `read_values` that takes one argument: the number of values to read. Since you are sending two values, you should call this with the argument 2. If you were sending three values from the Pico, you would call this with the argument 3, etc. Call this method in an infinite `while True` loop, e.g., `rec.read_values(2)`, and print the values out.

13. Now, let's run both programs! First, run `publish.py` *on the Pico!* Next, run `receive.py` on your laptop (use the play button at the top right, not in the bottom bar)! You should see the values being sent from the Pico printing in your terminal. Turn the potentiometers to make sure the values your laptop is printing change accordingly.

14. Finally, let's practice sending an additional value. In your `publish.py` file, add a third argument to the `pub.send()` call. For now, just send a constant value of your choice. IMPORTANT: remember, this value needs to be an int!

15. On the `receive.py` file, change your call to `rec.read_values()` to read three values instead of 2 and print out all three values.

16. Run your code again to confirm that you're now seeing three values sent and received instead of 2.

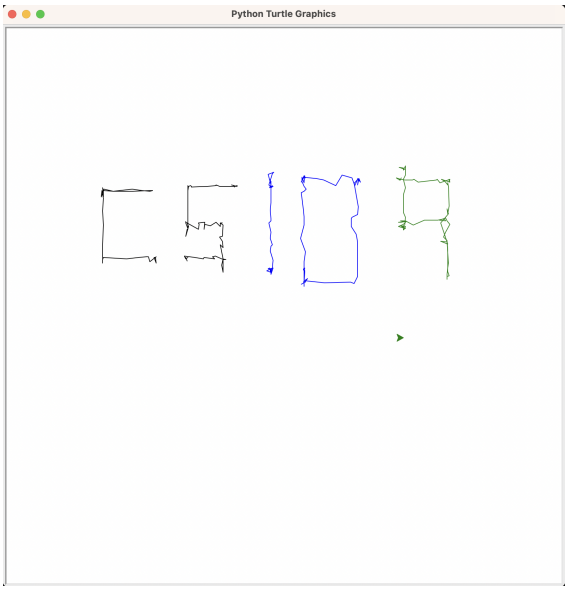


Figure 7: In the challenge, you'll use the potentiometers and buttons to control a Turtle graphics etch-a-sketch. Note, changing colors is optional. Also, it is normal for the output to be shaky.

Challenge: Build an Etch-a-Sketch

The challenge for this lab is to use the tools from the three explorations above to build an etch-a-sketch. The etch-a-sketch should consist of two programs:

1. `controls.py` is a program that runs on your Pico. It should read values from your potentiometers and at least one button and send values derived from them to your laptop, using the Publisher.

2. `etchasketch.py` is a program that runs on your laptop. It should setup a Turtle from turtle graphics, then sit in a while loop, listening to messages from your Pico using the Receiver class, and moving the turtle according to the potentiometer readings.

Most of the code for `controls.py` should come directly from your explorations.
You will, however, need to design a few pieces of logic for `etchasketch.py`.

1. The starter code provides a function called `setup_turtle(screen_width, screen_height)` that creates a Turtle, sets up a Turtle graphics window of a specific size in pixels and returns the Turtle object. Call this function in main to setup your drawing window. It's recommended to use a square window of size 800px by 800 px. Note that the center of the Turtle window is (0,0), which, if the window were 800x800 px would make the Turtle coordinates between -400 and +400 in both directions.

2. Your potentiometers each give you a reading between 0 and 65535. You should write a function called `pot2px(potvalue, min_px_val, max_px_val)` that maps the potentiometer reading to a Turtle coordinate (float) between a maximum and a minimum value.

   For example, `pot2px(0, -300, 600)` should return -300 and `pot2px(65535, 200, 400)` should return 400. Calling this function on a potentiometer reading between 0 and 65535 should return a float proportionally between `min_px_val` and `max_px_val`.

   You will use this function to translate potentiometer readings into drawing locations that fit into your window. Note that since the Turtle graphics origin is at the center of the drawing window, you'll likely always be calling this function with min and max pixel values that are symmetric around zero. However, you **should not assume that this is the case**.

In general, you may assume that `potvalue` is an `int` between 0 and 65535, that `min_px_val` and `max_px_val` are both ints, with `max_px_val` > `min_px_val`, and the difference between `max_px_val` and `min_px_val` being no greater than 2000.

Please do not change the input or output definitions of this function's behavior, as I will be testing this particular function.

I've included a program called `testpot2px.py` that you can use to test out your `pot2px` function. Note that these test cases may not be exhaustive.

3. Your program should read two potentiometer values from the `Receiver`, use `pot2px` to convert the left potentiometer to an x value and the right potentiometer to a y value, and then use the Turtle's `goto` method to send it to those coordinates.

4. Your program should also use a boolean flag variable which is controlled by a button on the Pico to turn the Turtle's pen on and off. In `controls.py`, add this state to the values you are sending to your laptop (remember to cast it to an int first, and remember to update the number of values you are receiving). Important: only call the send function once on all three values–do not split these into separate function calls. In `etchasketch.py`, use that value as a flag to turn your turtle's pen on and off.

Record a demo video of your working etch-a-sketch!

In your demo video, please use your etch-a-sketch to write "CS 109" and optionally draw something of your choice.

- Your writing/drawing should be large enough to show that you can draw over most of the Turtle window (i.e., don't draw in a small corner).
- Your drawing demo should demonstrate the pen-up pen-down functionality of your etch-a-sketch (e.g. between letters).
- Please verbally introduce your project before demo-ing it in your video.
- Submit your video demo on Canvas.

Writeup

As usual, there is a reflection to write and submit with the lab. Please complete the `writeup.md` template in the Github repository and push it with your code.

Rubric

The project will be graded as follows:

Functionality	Does the demo video clearly demonstrate the functionality of the etch-a-sketch, including drawing over the whole Turtle window using the two potentiometers, and turning the pen on and off with a button? Does `pot2px` correctly map between potentiometer readings and pixel values (and is not limited or hard-coded to only work for the specific pixel values used in the etch-a-sketch)?
Structure and Style	Is the code readable? Is there a reasonable decomposition into functions? Is the code needlessly repetitive? Are there excessive or unnecessary global variables? Does the program effectively use main as a single entry point? Do variables and functions have meaningful name? Are there docstrings on each file and function definition? Does the code have sufficient meaningful comments to document its functionality?
Writeup	Completeness: Are all the sections in the template completed? Clarity: Does the writeup clearly explain the project, how to use it, and correctly describe how it was implemented? Are there significant grammatical errors or typos that make the report difficult to understand? Ethical and Personal Reflections: Is there a thorough discussion of both benefits and risks of this kind of technology? Are personal reflections on the project discussed? Teamwork Does the contribution statement make it clear how both teammates contributed to the project and is the distribution of work even? This score and the overall lab score may be adjusted individually but group work issues should be brought to the professor BEFORE submission. Part of this score is completing and submitting the group contract.

Optional Extensions:

Here's some ideas that you might try out for fun. If you are planning to use any advanced programming constructs, you may only use them for an extension; please touch base with me first.

1. Add a button input to your controls and use it to update an int variable on the Pico representing a color. Add this int to the values you are sending to your laptop (remember to update the number of values you are receiving), and use that number to index an array of colors to set the turtle's color.

2. The output from the potentiometers is extremely unstable, resulting in shaky lines. Can you make the drawing more stable somehow?

3. Other ideas that you may have! Happy to chat about possibilities.