CS109 Lab 0: Intro to Physical Computing

Fall 2024

Overview

Learning Objectives

The purpose of this lab is to familiarize you with running Python code on the Raspberry Pi Pico microcontroller (MCU) while practicing the beginning programming concepts we've learned in class. By the end of the lab, you should be able to:

1. Interact with the Python interpreter on the Pico
2. Write and run a Python program on the Pico
3. Use the picozero modules classes and their methods to interact with components attached to the Pico

Prerequisites

Before coming to 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. Install the Micro Pico VSCode extension (see video from HW1 on Canvas)
4. Accept and clone the Lab 0 assignment from GitHub classroom on a computer that you bring with you to class.

Step 1: The Pico's Python Interpreter

1. Get a Lab Kit from the front of the classroom
2. Plug in the micro-USB cable to attach the Pico to your laptop. You may need to use a USB adapter. The Pico can draw power over USB, so plugging it in should power it on, although you may not see any visual indicators of this.
3. Open VSCode and open the folder where your Lab 0 repository is.
4. In VSCode, click on the MicroPico Icon in the left sidebar. At the bottom of the screen, you should see "Pico Connected" with a checkmark. If you don't, try unplugging and reseating the USB connections.
5. Open a new terminal by going to "Terminal–>New Terminal" in the top menu.
6. Once the terminal opens, click on the down arrow next to the plus near where it says "bash". Choose Pico (W) vREPL. REPL stands for Read, Evaluate, Print Loop. You should see a new terminal open with a prompt like the following:

```
MicroPython v1.22.2 on 2024-02-22; Raspberry Pi Pico W with RP2040
Type "help()" for more information or .help for custom vREPL commands.

>>>
```

Does this look familiar? Try out some Python commands. Record what works and what doesn't work below:

7. How do we know that this is all actually running on the Pico and not on your laptop? Let's do a quick experiment. The picozero module provides classes, functions, and variables that offer abstractions for a lot of things you might want to do with a device like the Pico. I've already installed the module for you on the Pico. Start by entering a command into the interpreter to import the class PicoLED from picozero. You should know how to do this–think about how you imported the Turtle class from the turtle module. Write the import statement you used below:

8. PicoLED is a class that controls a light that is built into the Pico board. We'll start by turning that light on and off. Create an instance of the PicoLED class you just imported and assign it to a variable called `pico_led`. Again, review your notes on working with the Turtle class if you're not sure how to do this. Write the statement you used below:

9. The PicoLED class exposes the following instance variables and methods (not an exhaustive list):

Instance Variable		Description
value		Current state of the device, e.g. on or off
Method	**Parameters**	**Description**
on	None	Turns device on
off	None	Turns device off
toggle	None	Sets the device to the opposite state

   Looking at this table, can you figure out how to turn the LED on (there is more than one way)? Set the appropriate value or call the appropriate method in the interpreter. Don't forget to reference your PicoLED object, `pico_led`. Write the command you used below. Does the LED light up? Can you enter a few commands to turn it on and off?

Step 2: Running a Python Program on the Pico

Of course, it's not that useful to manually turn devices on and off. Let's write a simple Python program to turn the LED on and off three times instead.

1. Open the file called `blink.py` in your Lab 0 repository. Let's start simple–write a series of Python statements to create a PicoLED object and turn it on once. Don't forget your import statement!

2. When your program is ready, look at the very bottom of your VS Code window for a play button that says Run. It should be next to a button that says Pico connected. Hit the button. Does your LED turn on?

3. Great! Now, add a few more lines to your program to turn the LED on and off three times in a row. Run it again. What happens?

4. It turns out that the Pico processed the instructions in your program so quickly that it cycled your LED on and off before you could see it happen. Let's slow it down so that we can observe it changing. To do this, we'll use a function called sleep() from the time module. Start by importing sleep from time at the top of your program.

5. The sleep() function takes one argument: the number of seconds that you want to delay. When you call it, it essentially pauses the execution of your program for the length of the delay you specified, then continues. Try calling this function to add delays so that the LED turns on for 1 second, then off for one second, and so on, over the three times you cycle it. Hint: you will need to call the sleep function multiple times. Run your program on the Pico again. What happens this time?

6. Before you go on, add some comments to your code to explain what it is doing.

Step 3: Connecting an Output Device–The Buzzer

Next, we'll learn how to play a tone using the buzzer that's attached to the prototyping board your Pico is seated on (it's the black cylinder in the bottom left). To do this, we'll have to connect the buzzer to one of the *pins* on the Pico. These are the metal (conductive) studs sticking out of the pico–on your prototyping board, each one is labeled.

1. Find the pin that says `GP5` and slip one end of a F-F hookup wire over it, like in the figure.
2. Now, find the pin that says `BEEP` elsewhere on the board. Attach the other end of the wire to this pin. The buzzer is *not* attached to the Pico by default–by connecting a wire between the buzzer pin and pin `GP5` on the Pico, you've just connected the buzzer to the Pico, via the Pico's pin number 5.
3. Now that the buzzer is connected, let's import the Buzzer class from the picozero module and play around with it! To create a Buzzer object connected to the pin you attached the wire to (pin 5), pass the pin number as an argument to the constructor when you call it, e.g. `b = Buzzer(5)` for pin 5.

   Here's some of the instance variables and methods for the Buzzer class:

Instance Variable		Description
value		Current state of the buzzer, e.g. playing or not
Method	**Parameters**	**Description**
on	None	Turns the buzzer tone on
off	None	Turns the buzzer tone off
beep	`on_time`, `off_time`, n	Turns the buzzer on and off, for `on_time` and `off_time` seconds, n times. If n is set to None, then it will beep repeatedly without stopping.

   Try these out in the interpreter–make sure you understand how to turn the buzzer on and off.

4. Next, open up the file `beep.py` and write a short program that turns the buzzer on and off three times. This time, make the length of the tone and the pause between tones decrease, from 2 seconds to 1 second to 0.5 seconds.
5. Comment your code before continuing.

Challenge: Morse Code Generator

Morse code is a specification of beeps used to send messages. For example, "SOS" is a famous distress signal indicated by three short beeps (called dots) followed by three long beeps (called dashes) followed by three short beeps (called dots). Morse code was also used to send messages over telegraph lines.

Each letter in the English alphabet has an associated pattern of dots and dashes. In order to spell words and sentences, a Morse code operator can string together a series of letters using the following rules:

1. A dot is a beep that lasts for 1 time unit
2. A dash is a beep that lasts for 3 time units
3. Pause 1 time unit between the beeps that make up a letter
4. Pause 3 time units between letters
5. Pause 7 time units between words

For example, if we wanted to send the message "SOS SOS", we would emit the following beep sequence:

- dot (pause 1) dot (pause 1) dot (pause 3)

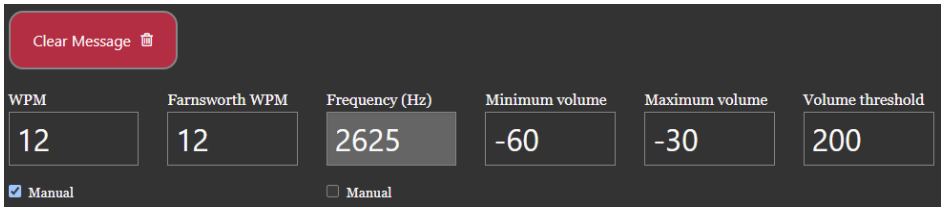

Figure 1: Use these settings in the Morse Code decoder.

- dash (pause 1) dash (pause 1) dash (pause 3)
- dot (pause 1) dot (pause 1) dot (pause 7)
- dot (pause 1) dot (pause 1) dot (pause 3)
- dash (pause 1) dash (pause 1) dash (pause 3)
- dot (pause 1) dot (pause 1) dot (pause 7)

For the final piece of this introductory lab, your challenge is to write a Python program, in the file called `morse.py`, that sends a message using Morse code with the buzzer.

You can play around with the time unit–in my case, I found that a time unit of about 0.1 seconds seemed to work well with the online Morse code decoder below.

Your program should meet the following specifications:

1. Your message must consist of at least three words, which use at least three different letters across them.
2. You should test it by playing it to this online Morse code decoder: `https://morsecode.world/international/decoder/audio-decoder-adaptive.html` Important: try it out with the settings below (you need to set them to manual). You may need to tweak these to get the detector to detect your specific buzzer with your specific time unit, but the below settings have worked well in testing.
3. Your code should apply best practices in terms of structure and style. For example, use variables instead of magic numbers, give your variables self-documenting names, and use comments to explain what your code is doing.
4. You should not use any programming constructs that we have not yet covered in class or in the reading. In particular, do not use loops, functions, or conditional statements.

Submission

You will submit the following:

1. Record a video demonstration of your Morse Code generator. It should be clear what you are demonstrating in the video; please verbally describe what you are doing in the video. Your video should demonstrate that it works–the best way to do this would be by playing your message and showing it correctly detect using the online decoder linked above. You will submit this video on Canvas.
2. Commit and push your final code in `morse.py`. Again, I will be looking for good structure and style in this code, as well as sufficiently clear comments to describe your solution.
3. Also complete the written reflection in `reflection.md`.