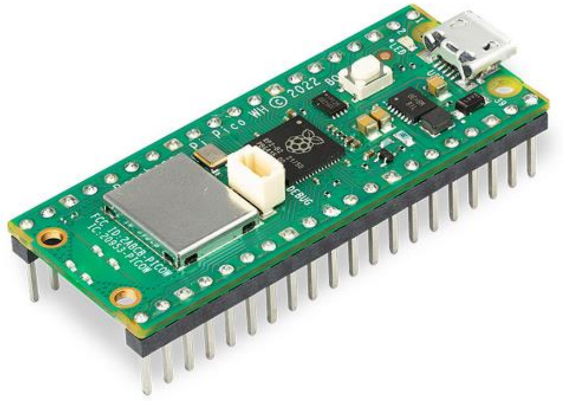
Introduction to Raspberry Pi Pico Wireless Educational Kit

GK – EK – PICO

Table of Contents

Brief Overview

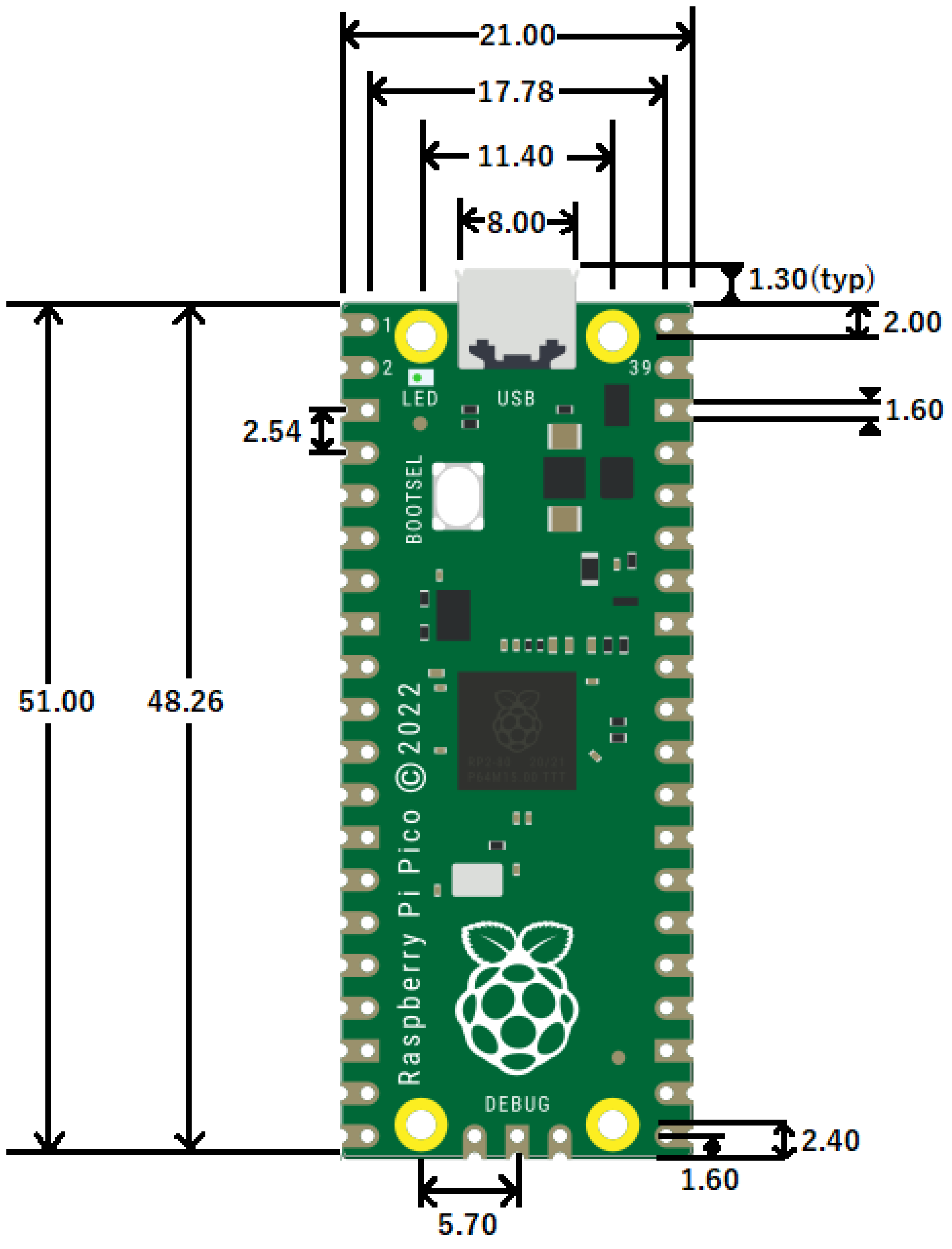
Raspberry Pi Pico is a high-performance, low-cost microcontroller with digital interfaces. It has a dual-core Arm Cortex M0+ processor capable of running at up to 133MHz, containing 264KB of SRAM, and 2MB of flash memory. For programming, Raspberry Pi's C/C++ SDK and MicroPython are available.



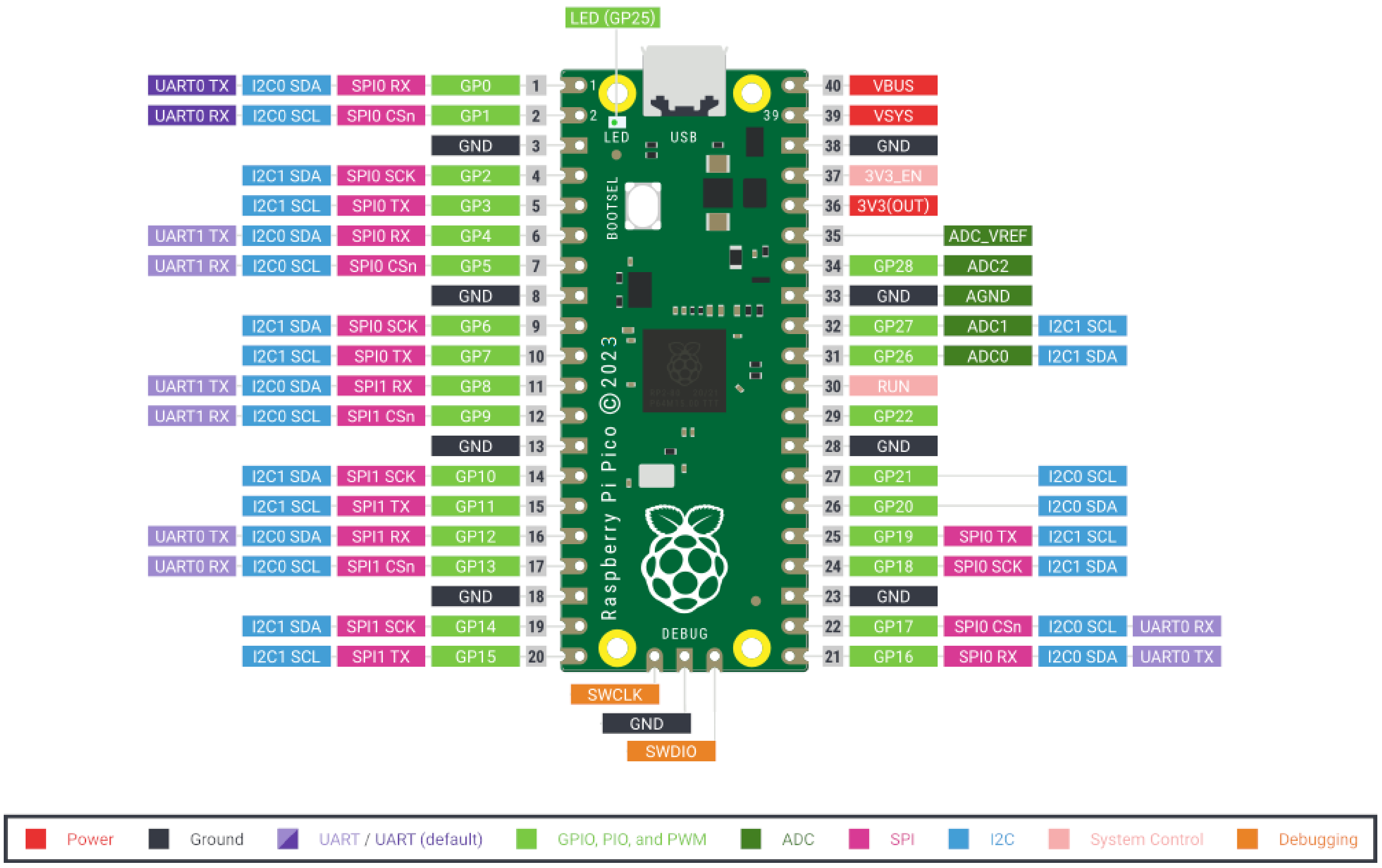
Raspberry Pi Pico Features

- RP2040 MCU chip.
- Dual-core Arm Cortex M0+ processor, the flexible clock running at up to 133MHz.
- 264KB of SRAM, 2MB of Flash memory.
- USB 1.1 with host and device support.
- Low-power sleep mode.
- Programming using storage over USB.
- 26 flexible GPIO pins.
- 3x 12-bit ADC, 2x SPI, 2x I2C, 2x UART, 16x PWM channels.
- An accurate clock and timer.
- A temperature sensor.
- 8x Programmable I/O state machines for custom utilization of peripherals.

Dimensions


21.00
17.78
11.40
8.00
1.30(typ)
2.00
1.60
2.54
51.00
48.26
2.40
1.60
5.70
1
2
39
LED
USB
BOOTSEL
Raspberry Pi Pico ©2022
DEBUG

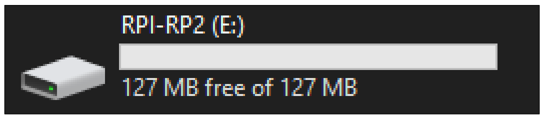
Pinout



Note: If the debug pins in your Pi Pico are located near the middle of the board, they are in the same order as shown in the diagram above.

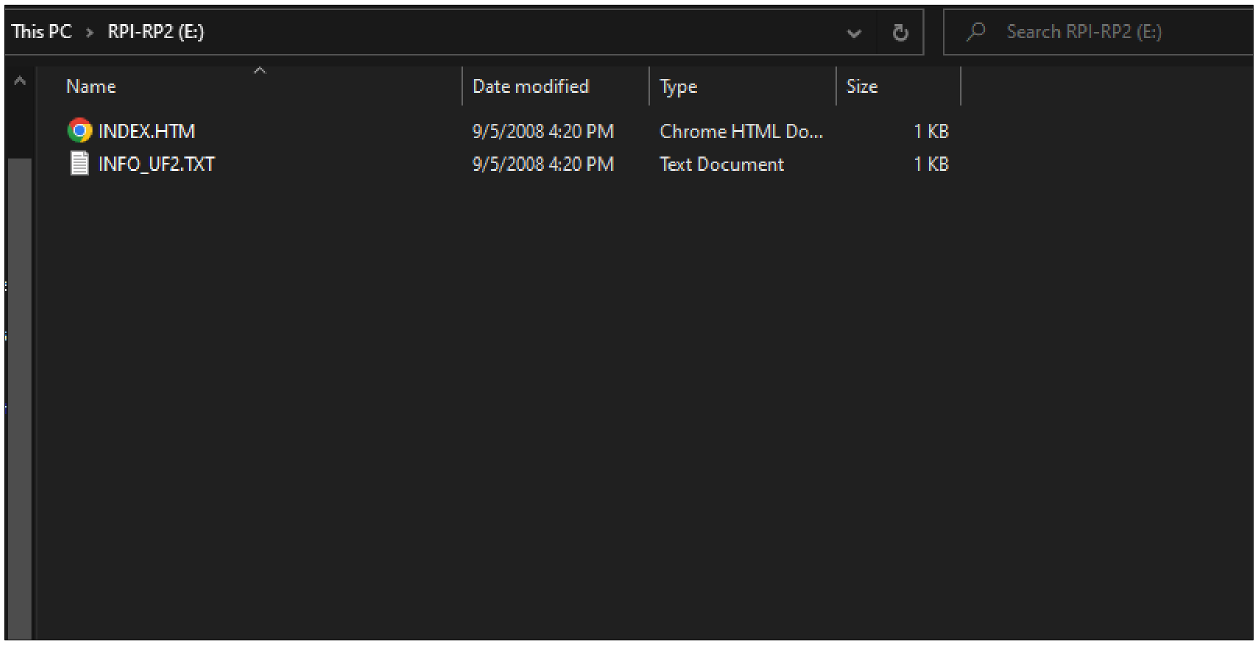
Firmware Installation

To install the firmware on your Pi, you shall first download its file here. Connect your Pi Pico to your computer.



If you do not see the above drive appear under **Devices and drivers** in **This PC**, try plugging in the Pi Pico while pushing down on the **BOOTSEL** button.

When you access the drive, just paste the file downloaded in there.



Name	Date modified	Type	Size
INDEX.HTM	9/5/2008 4:20 PM	Chrome HTML Do...	1 KB
INFO_UF2.TXT	9/5/2008 4:20 PM	Text Document	1 KB

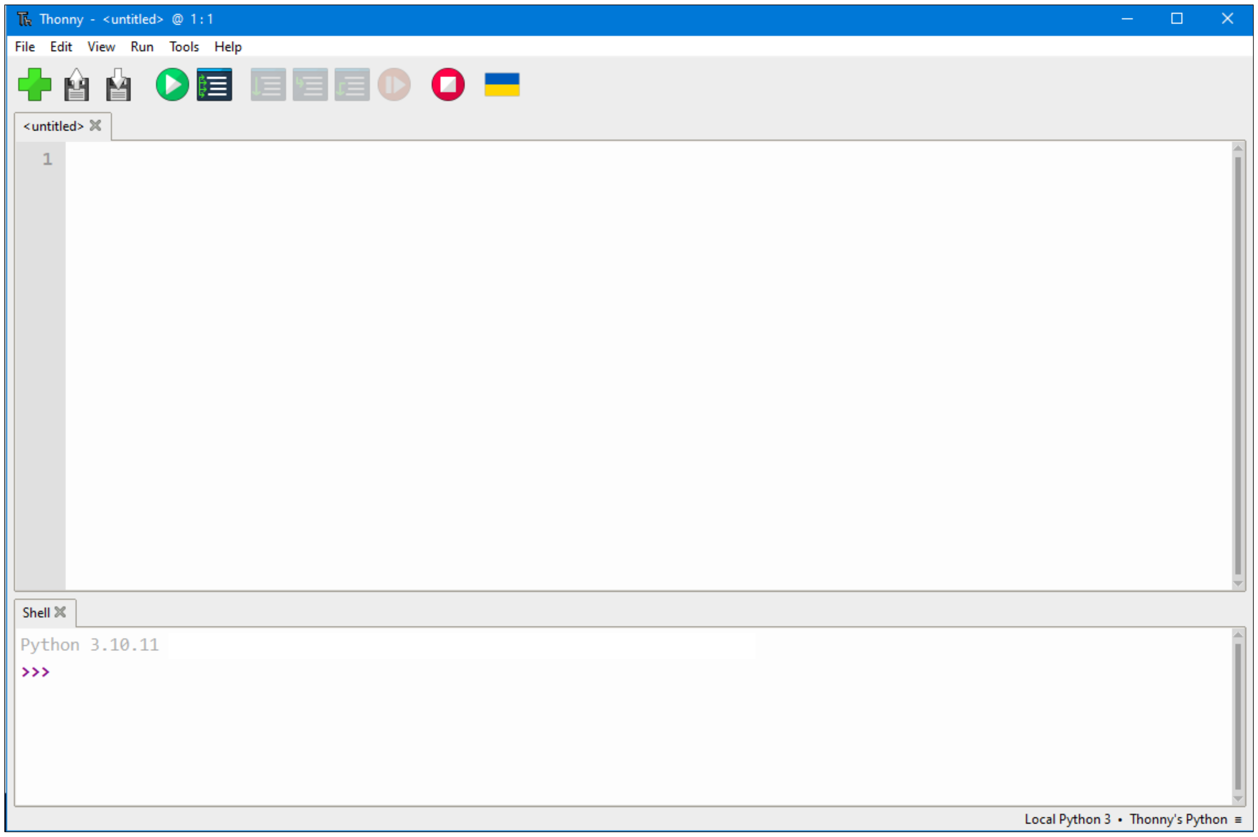
When the transfer is complete, the window will close automatically and the Pi Pico will eject itself from your computer. If you'd like to remove the firmware, connect the Pi to the computer while **BOOTSEL** is pushed down.

Intro to Thonny

Thonny comes with built-in Python and is the perfect installation for beginners thanks to its friendly user interface, capability of highlighting syntax errors, and having a simple debugger. Overall, the program is deprived of features that may distract beginners.

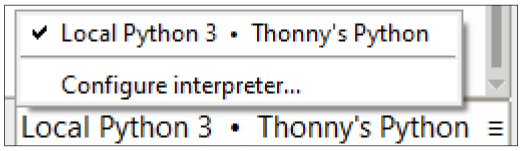
To begin programming on your Pi Pico to perform the exercises in this document and get familiar with coding, you will need to download Thonny here. Install the version that suits your operating system on the official Thonny website.

When launching it, you will be greeted with a pop-up window where you will need to specify preferred language and initial settings (standard will suffice). Click **Let's go!** The following window will pop up next:

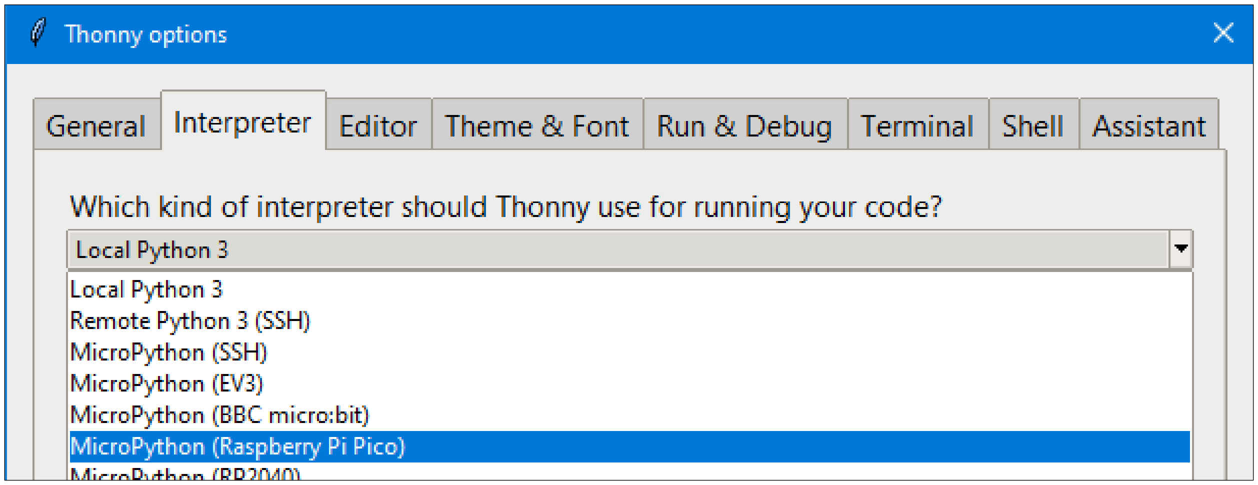


```
Python 3.10.11
>>>
```

Moving on, now you must configure the Python environment and select the Pico port. To do this, you must first connect your Raspberry Pi Pico to your computer if it isn't already. Next, located on the very bottom right of the window, you must click where it says **Local Python 3 • Thonny's Python**. The exact text may vary with the version of Thonny, but you are essentially going to be selecting **Configure Interpreter…**


✔ Local Python 3 • Thonny's Python
Configure interpreter...
Local Python 3 • Thonny's Python ≡

Another window will pop up that displays Thonny options. With the **Interpreter** settings tab already selected, click the Interpreter dropdown menu and find **MicroPython (Raspberry Pi Pico)**. Select it.

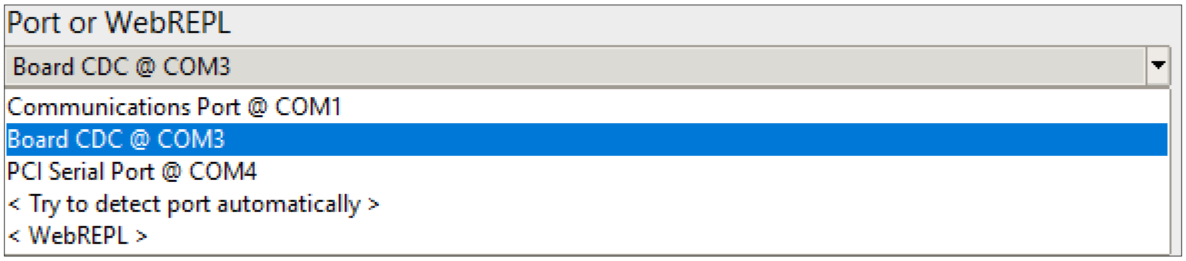


Which kind of interpreter should Thonny use for running your code?

Local Python 3

- Local Python 3
- Remote Python 3 (SSH)
- MicroPython (SSH)
- MicroPython (EV3)
- MicroPython (BBC micro:bit)
- MicroPython (Raspberry Pi Pico)
- MicroPython (RP2040)

The window will change and you will be asked to select a port or WebREPL. Click the dropdown menu and find the port that the Pi Pico is connected to. Name and COM number may vary from device to device.

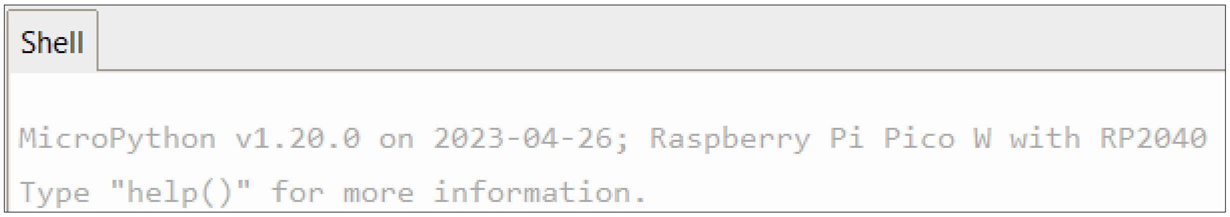


Port or WebREPL

Board CDC @ COM3

- Communications Port @ COM1
- Board CDC @ COM3
- PCI Serial Port @ COM4
- < Try to detect port automatically >
- < WebREPL >

Click **OK** to confirm. The text in the Shell window should now change to the accustomed Pi device:



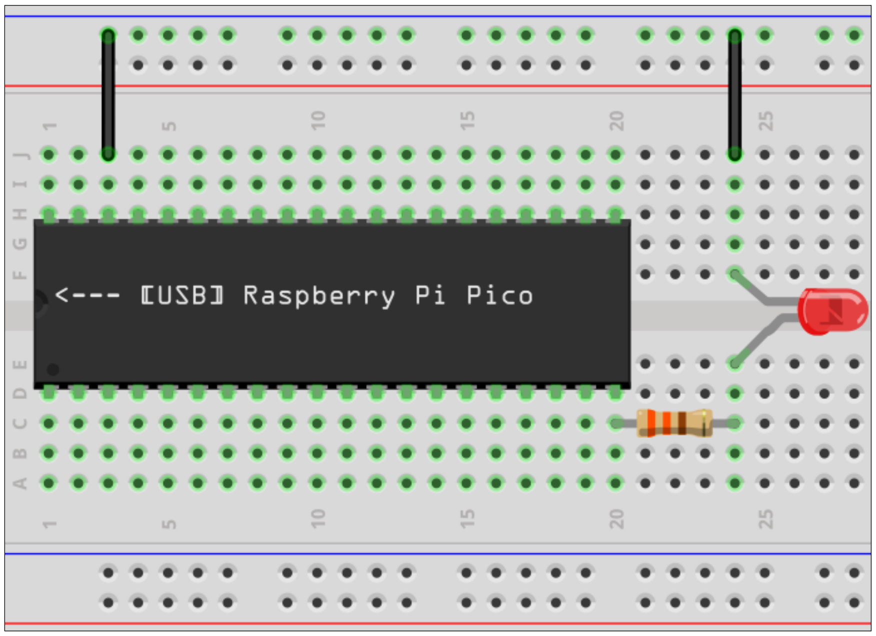
```
MicroPython v1.20.0 on 2023-04-26; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
```

Mini Test

When the pico is ready to be programmed, paste this code into Thonny but do not run it yet because nothing is going to happen.

```
import time


led_external = machine.Pin(15, machine.Pin.OUT)#Set GPIO pin 15
#as the output pin and define the LED with it.

while True:

    led_external.toggle()

    time.sleep(2) #A delay of two seconds.
```

Fetch your breadboard and connect the following components as shown in the diagram below:

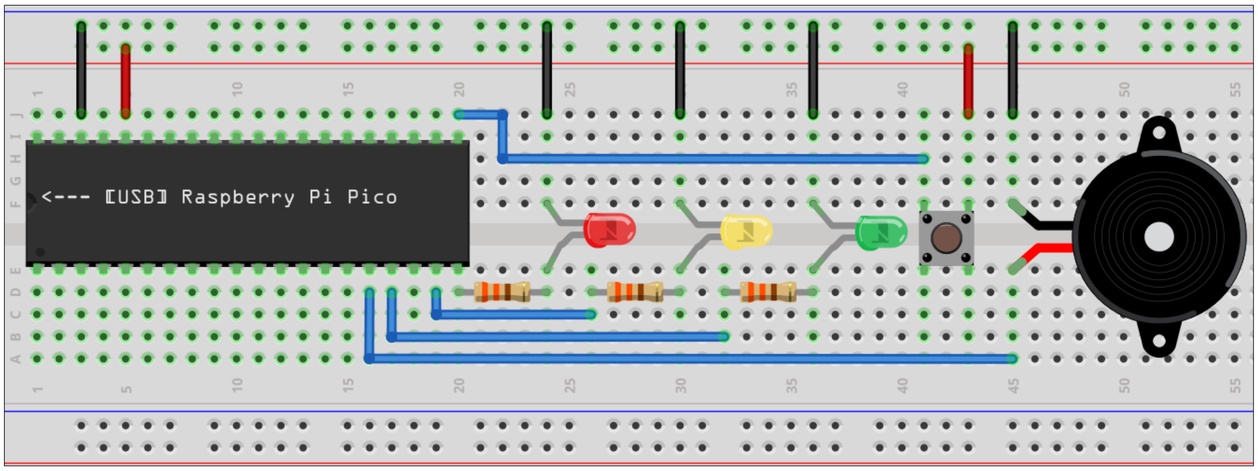


Just a friendly reminder to pay attention to how you place the LED. Remember – the flat side and shorter pin indicate that it's the ground pin! Also, bear in mind that an LED should never be plugged in without a resistor unless it has one built-in.

Now push the code by pressing: ▶

If you see your LED blink at a frequency of 0.5Hz (2 seconds), congratulations! Your Raspberry Pi Pico is ready for the exercises.

Exercise 1: Traffic Lights System



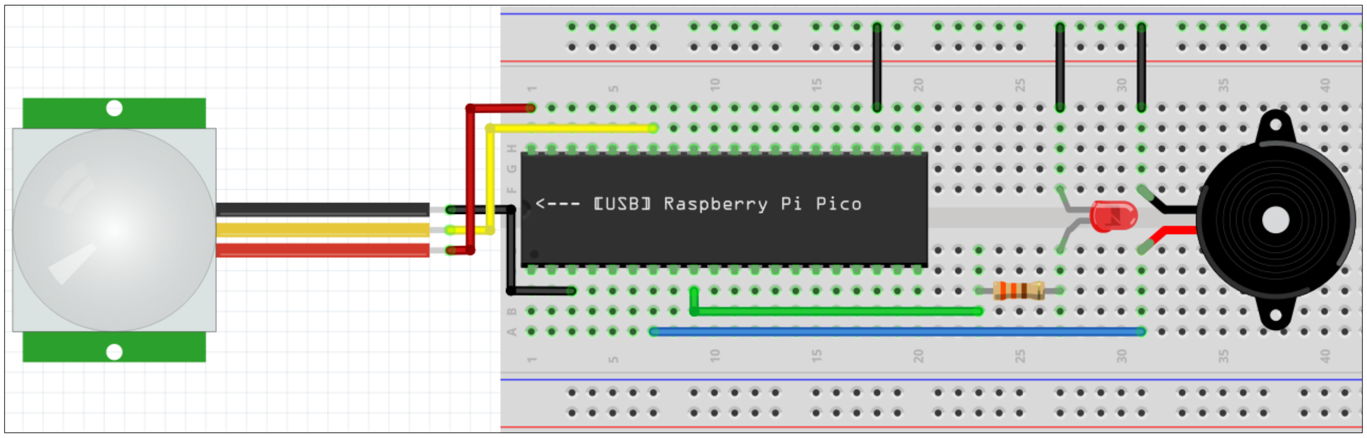
Connect the breadboard as specified in the above diagram. If you're having trouble plugging the buzzer's unshrouded wires into the breadboard, try twisting each end into a tight knot.

Open the **Exercises** folder you have downloaded and open **Exercise_1.py**. Push the code to the Raspberry Pi and observe the LEDs. You can see that the LEDs light up in the same order as the traffic lights do, simulating their process. You can see the delays in the code and when the corresponding LED has a value of 1 to turn it on and a value of 0 to turn it off.

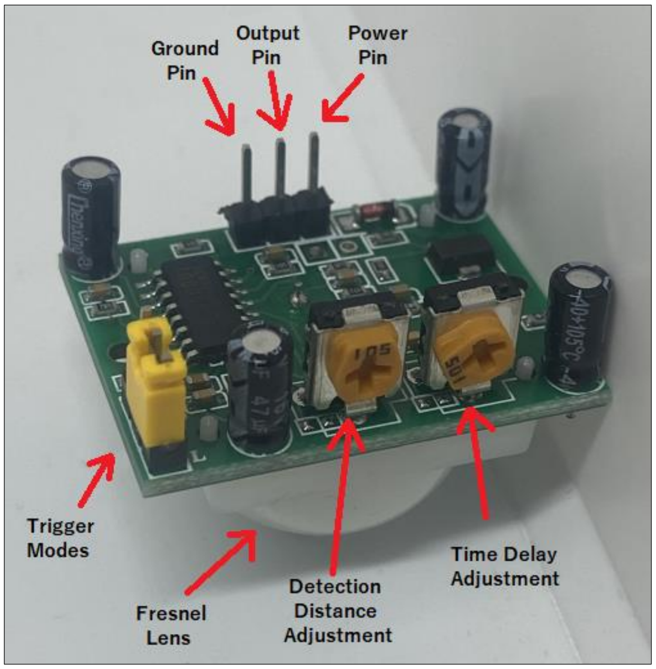
By pressing the button, the program uses a thread to remember that the button has been pressed once and waits until the LEDs finish executing before returning to the if statement at the very beginning of the main `while True` loop. The buzzer buzzes once and the program makes sure to mark the button as not pressed anymore. This `if` statement could be useful for making the delay of the red LED much longer to allow some separate pedestrian LED to light up so they can cross. If you'd like, you can make that system up yourself!

Task: Add a fourth LED (green) (do not forget the resistor!) to turn on for 5 seconds and turn off afterwards when the if statement in the main `while True` loop executes. Hint: Look at how the other LEDs were defined and utilized, as well as use the time.sleep() function. Be sure to define the LED with a GP pin.

Exercise 2: LED & Buzzer Burglar Alarm



Connect the breadboard as specified in the above diagram. Refer to the image below to understand the fundamentals of the passive infrared sensor.



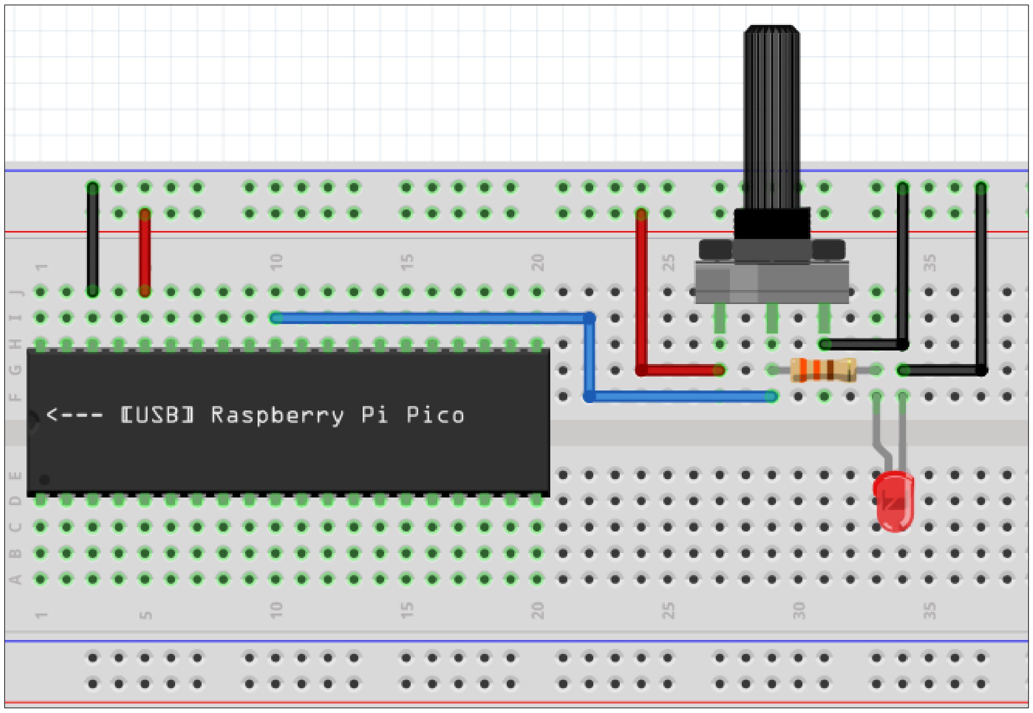
This sensor has two trigger modes: B mode is used for a repeatable trigger where the LED stays on when there is motion detected, and L mode is non-repeatable where the LED blinks at a rate of 1 second when there is activity detected.

The detection distance potentiometer can be set to detect motion between approximately 2 – 10 meters.

Open **Exercise_2.py** and push the code to the Pi Pico. Move your hand in front of the Fresnel lens and the LED should blink 3 times with the buzzer. Notice how there is just a simple 1ms delay in the main loop – that's because the program requires some degree of activity.

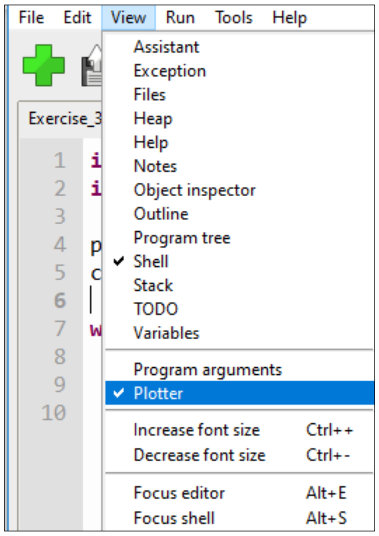
Task: Decrease the detection distance significantly and make the LED blink one time with the buzzer going off once. Hint: Refer to the image and the `for` loop in the code.

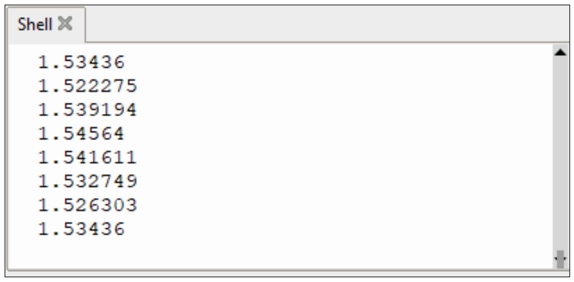
Exercise 3: Reading the Potentiometer



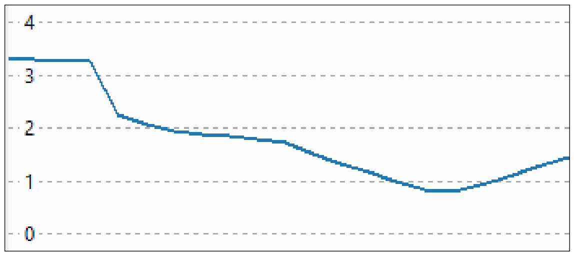
Connect the breadboard as specified in the above diagram. Open the **Exercises** folder you have downloaded and open **Exercise_3.py**. Push the code to the Raspberry Pi.

Navigate to the tabs on top of the Thonny application and select **View -> Plotter**. A graph will be shown to display the current voltage passing through the potentiometer to light up the red LED.





```
1.53436
1.522275
1.539194
1.54564
1.541611
1.532749
1.526303
1.53436
```

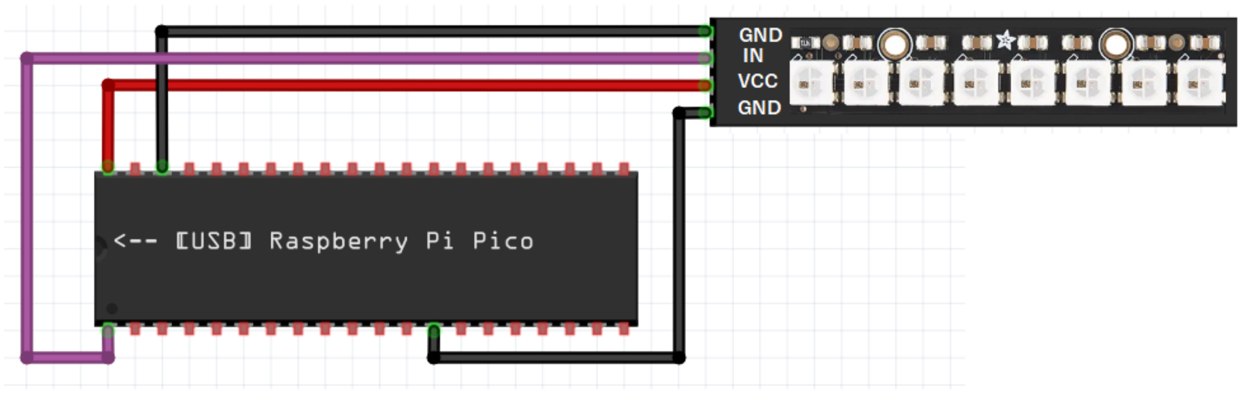


However, the values would not be displayed as voltages if not for the conversion factor of 3.3 / 65535. Otherwise, you'd only see raw values. Try moving the potentiometer to see how it affects the values and the brightness of the LED.

Task: Implement a second LED (do not forget the resistor!) that turns on when the voltage passing through the first LED is greater than or equal to (>=) 3 volts and turns off when less than (<) 3 volts. Hint: Use the `if` statement pseudocode established below to help you out. Refer to Exercise 1 to set up the LED if you must.

```
if voltage greater than or equal 3:
     LED is ON
else:
     LED is OFF
```

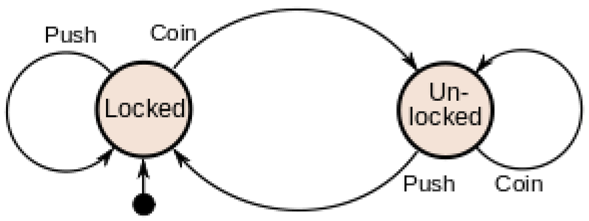
Exercise 4: The WS2812-8



A breadboard is optional for this exercise. Connect the Pi Pico and the WS2812-8 as shown in the above diagram and open **Exercise_4.py** on Thonny. Run the code.

Upon running the code, you can see all eight LEDs on the WS2812-8 flash three different colors with a delay in between. Because this is one big component, it's a bit more complicated than just a regular LED and requires more lines of code to configure and set up. Because of that, all three alternating colors are put inside a **state machine** system. A state machine is essentially just a big loop in a program or system consisting of states that transfer from one another. A transfer can happen through delays or events.

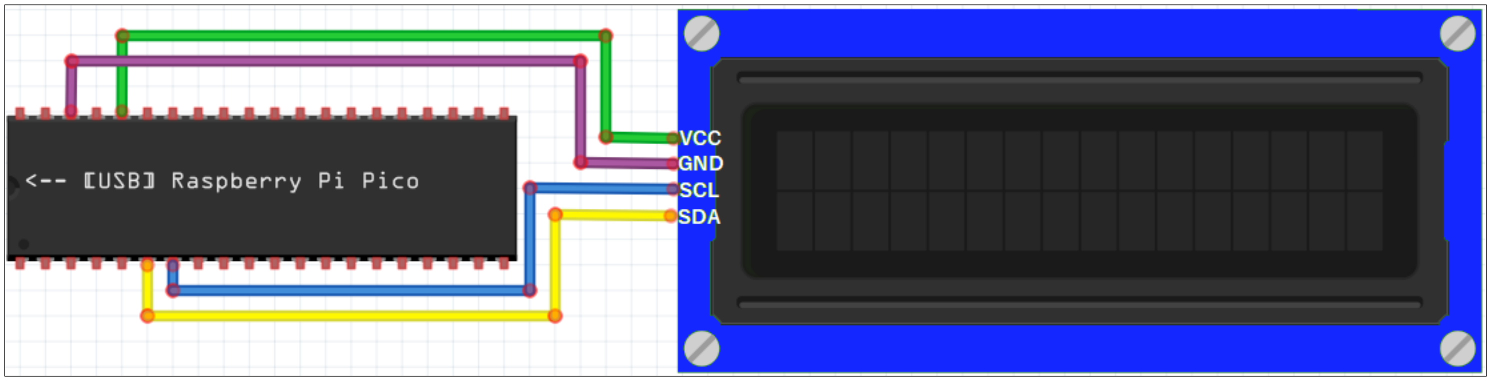
Visually, a state machine will look like this:



This example uses a coin lock system. When a lock is locked and you try pushing it, it will not unlock. However, when you provide a coin, the lock unlocks (state transfer). Providing another coin will keep the lock unlocked. When you push the unlocked lock, it will lock (state transfer). This whole procedure makes up this state machine system. "Locked" and "Unlocked" are the states, while "Push" and "Coin" are its triggers.

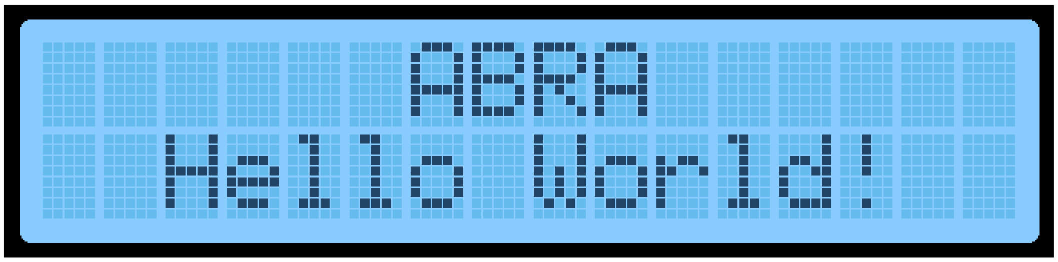
Task: Draw the state machine for this exercise's code. Hint: the three alternating colors are the program's states and the 5ms delays are its triggers.

Exercise 5: LCD1602 I2C Display



A breadboard is optional for this exercise. Connect the Pi Pico and the LCD1602 as shown in the above diagram and open **RGB1602.py** on Thonny. Save that file onto the Pi Pico. Open **Exercise_5.py** and run the code.

You should see the following text pop up on the LCD screen:



The background color should also be changing as the text is displayed. The variables "r", "g", and "b" are responsible for the colors and as you can see, each of the three color channels are composed of a formula. Each formula contains a variable named "t" that changes with each loop, causing the whole color to change.

Reading further, you can see the functions that are responsible for the text on the LCD screen.

Task: Change the first line of the LCD screen so instead of "ABRA" it says your first name. Center it. If your first name contains an odd number of letters, do not worry – add an exclamation point! Hint: Use the space key to manipulate how many rows to the right the text is.

Congratulations!

You have finished the five fundamental exercises and learned how to use many peripherals with the Raspberry Pi Pico!