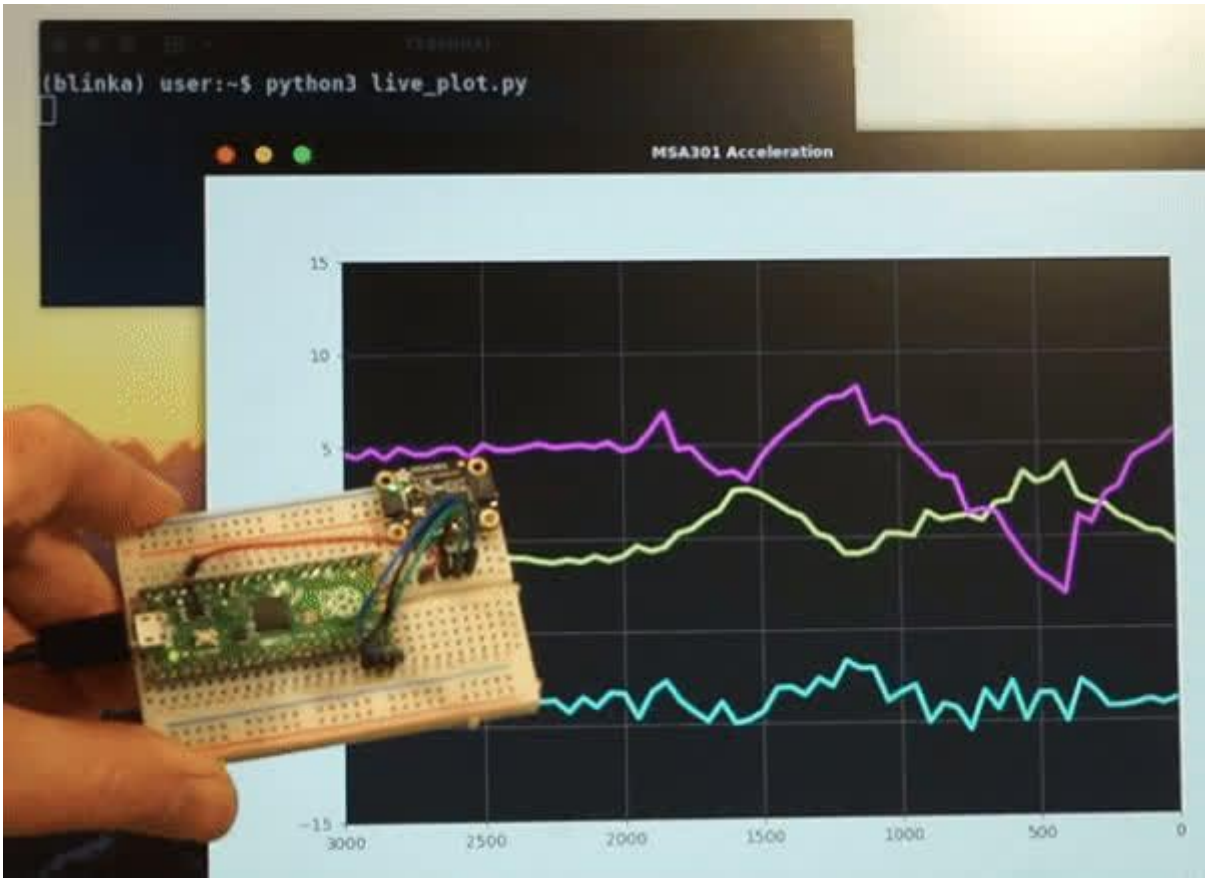
CircuitPython Libraries on any Computer with Raspberry Pi Pico

Created by Carter Nelson

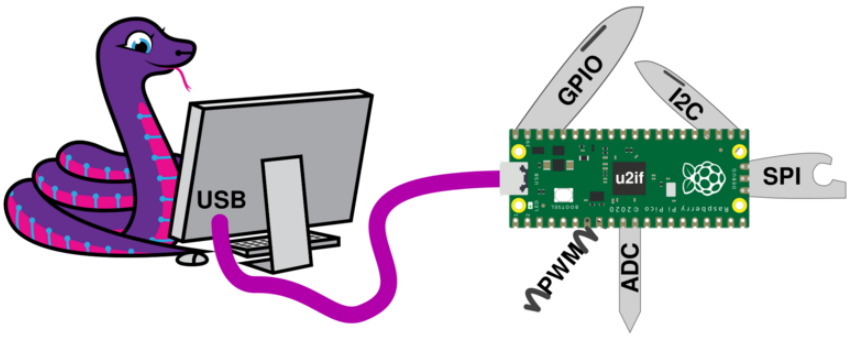


https://learn.adafruit.com/circuitpython-libraries-on-any-computer-with-raspberry-pi-pico

Last updated on 2023-07-07 01:38:08 PM EDT

Table of Contents

Overview



This guide will show you how to use a Raspberry Pi Pico RP2040 to connect various sensors and breakouts to your PC running Windows, Mac OSX, or Linux. Special firmware gets loaded onto the Pico and turns it into a sort of Swiss army knife providing:

- General Purpose digital Input and Output (GPIO) for things like buttons and LEDs
- Analog to Digital Conversion (ADC) for reading analog signals
- Pulse Width Modulation (PWM) for servos or LED dimming
- I2C and SPI for connecting *lots* of external sensors, displays, etc.
- NeoPixels (WS2812B) for happy rainbow blinky fun!

This is very similar to what the FT232H () and MCP2221 () already provide.

The u2if firmware is considered "experimental".

The approach in this guide is useful if you want to run "regular" Python code on your main computer and have it communicate with external devices connected through the Pico (or other RP2040 board). If you are instead trying to run MicroPython code directly on the Pico and use CircuitPython libraries, then see this other guide: CircuitPython Libraries on MicroPython using the Raspberry Pi Pico ().

The Magical u2if Firmware

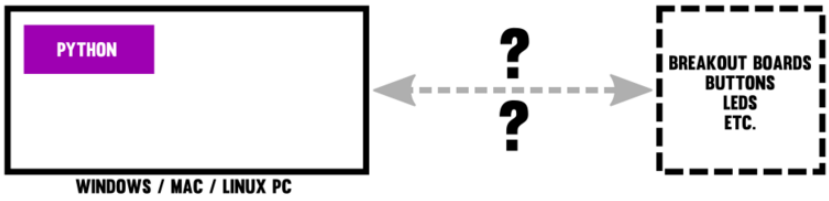
The key element to enabling this capability on the Raspberry Pi Pico is thanks to the excellent u2if firmware () written by execuc (). The main repo not only contains the firmware that goes on the Pico itself, but micropython compliant Python code for interfacing to the Pico from your PC. So if you're more used to the micropython interface, then checkout the u2if repo. It has everything you need.

u2if project

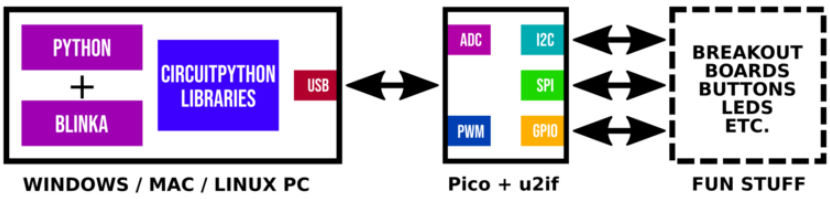
In this guide, we use the exact same firmware on the Pico. But on the PC, we use the newly updated Blinka library which has added support for interfacing with a Pico running the u2if firmware.

CircuitPython Libraries on Personal Computers

This is essentially the same idea as discussed in the FT232H Guide () and the MCP2221 Guide (). How can we directly connect common hardware items like buttons and LEDS (GPIO) or sensor breakouts (I2C/SPI) to a PC?



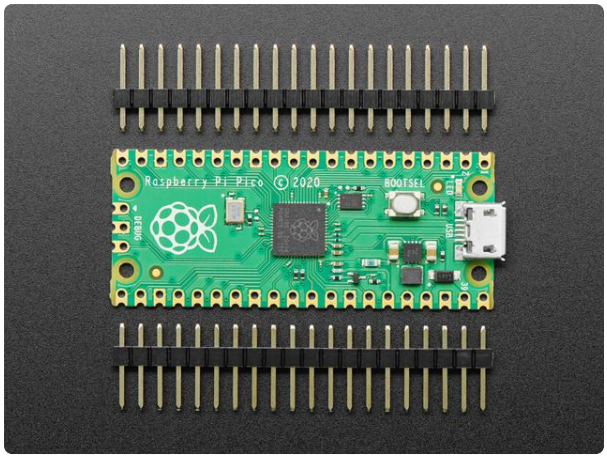
By loading the u2if firmware () onto the Pico, it turns it into sort of a bridge using USB on the main PC. So you end up with something like this:



On the computer, we install Blinka which provides a CircuitPython compliant interface to the Pico with u2if. That way, all the CircuitPython libraries can then be used - on your PC!

Required Hardware

The main requirement is the Raspberry Pi Pico RP2040.



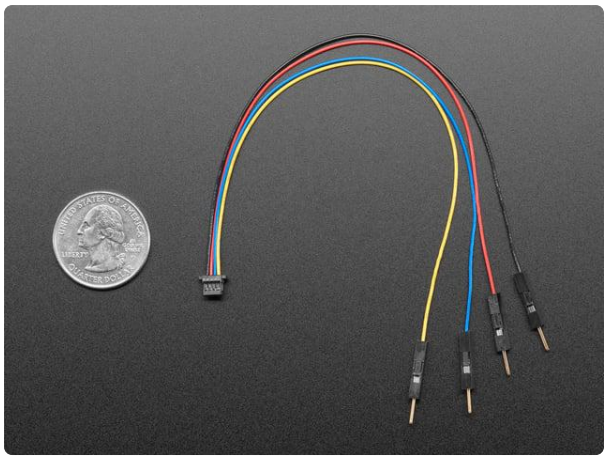
[Raspberry Pi Pico RP2040 with Loose Unsoldered Headers]
The Raspberry Pi foundation changed single-board computing when they released the Raspberry Pi computer, now they're...
https://www.adafruit.com/product/4883

You'll need a USB cable for programming and interacting with the Pico - but you probably have one of these laying around. Just make sure it's not a charge only cable.

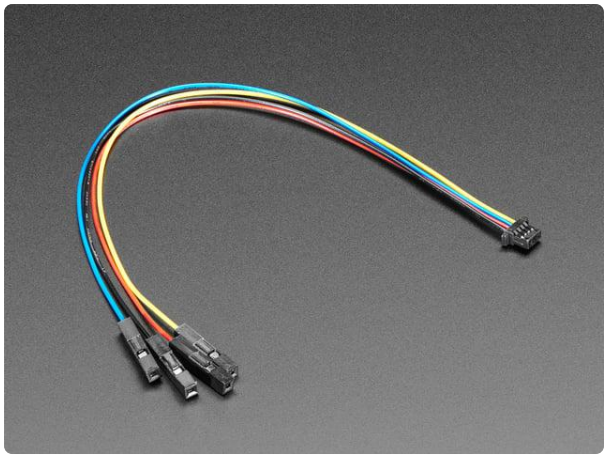
Beyond that, it all depends on what you want to do. There are examples provided later in this guide that show some typical use cases.

Other Hardware

If you're using STEMMA QT breakout boards, these cables can be helpful.



[STEMMA QT / Qwiic JST SH 4-pin to Premium Male Headers Cable]
This 4-wire cable is a little over 150mm / 6" long and fitted with JST-SH female 4-pin connectors on one end and premium Dupont male headers on the other. Compared with the...
https://www.adafruit.com/product/4209



[STEMMA QT / Qwiic JST SH 4-pin Cable with Premium Female Sockets]
This 4-wire cable is a little over 150mm / 6" long and fitted with JST-SH female 4-pin connectors on one end and premium female headers on the other. Compared with the chunkier...
https://www.adafruit.com/product/4397

Running CircuitPython Code without CircuitPython

There are two parts to the CircuitPython ecosystem:

- CircuitPython firmware, written in C and built to run on various microcontroller boards (not PCs). The firmware includes the CircuitPython interpreter, which reads and executes CircuitPython programs, and chip-specific code that controls the hardware peripherals on the microcontroller, including things like USB, I2C, SPI, GPIO pins, and all the rest of the hardware features the chip provides.
- CircuitPython libraries, written in Python to use the native (built into the firmware) modules provided by CircuitPython to control the microcontroller peripherals and interact with various breakout boards.

But suppose you'd like to use CircuitPython libraries on a board or computer that does not have a native CircuitPython firmware build. For example, on a PC running Windows or macOS. Can that be done? The answer is yes, via a separate piece of software called Blinka. Details about Blinka follow, however it is important to realize that the CircuitPython firmware is never used.

CircuitPython firmware is NOT used when using Blinka.

Adafruit Blinka: a CircuitPython Compatibility Library

Enter Adafruit Blinka. Blinka is a software library that emulates the parts of CircuitPython that control hardware. Blinka provides non-CircuitPython implementations for `board`, `busio`, `digitalio`, and other native CircuitPython modules. You can then write Python code that looks like CircuitPython and uses CircuitPython libraries, without having CircuitPython underneath.

There are multiple ways to use Blinka:

- Linux based Single Board Computers, for example a Raspberry Pi
- Desktop Computers + specialized USB adapters
- Boards running MicroPython

More details on these options follow.

Raspberry Pi and Other Single-Board Linux Computers

On a Raspberry Pi or other single-board Linux computer, you can use Blinka with the regular version of Python supplied with the Linux distribution. Blinka can control the hardware pins these boards provide.

Desktop Computers

On Windows, macOS, or Linux desktop or laptop ("host") computers, you can use special USB adapter boards that that provide hardware pins you can control. These boards include MCP221A () and FT232H () breakout boards, and Raspberry Pi Pico boards running the u2if software (). These boards connect via regular USB to your host computer, and let you do GPIO, I2C, SPI, and other hardware operations.

MicroPython

You can also use Blinka with MicroPython, on MicroPython-supported boards (). Blinka will allow you to import and use CircuitPython libraries in your MicroPython program, so you don't have to rewrite libraries into native MicroPython code. Fun fact - this is actually the original use case for Blinka.

Installing Blinka

Installing Blinka on your particular platform is covered elsewhere in this guide. The process is different for each platform. Follow the guide section specific to your platform and make sure Blinka is properly installed before attempting to install any libraries.

Be sure to install Blinka before proceeding.

Installing CircuitPython Libraries

Once Blinka is installed the next step is to install the CircuitPython libraries of interest. How this is down is different for each platform. Here are the details.

Linux Single-Board Computers

On Linux single-board computers, such as Raspberry Pi, you'll use the Python `pip3` program (sometimes named just `pip`) to install a library. The library will be downloaded from pypi.org () automatically by `pip3`.

How to install a particular library using `pip3` is covered in the guide page for that library. For example, here is the `pip3` installation information () for the library for the LIS3DH accelerometer.

The library name you give to `pip3` is usually of the form `adafruit-circuitpython-libraryname`. This is not the name you use with `import`. For example, the LIS3DH sensor library is known by several names:

- The GitHub library repository is Adafruit_CircuitPython_LIS3DH ().
- When you import the library, you write `import adafruit_lis3dh`.
- The name you use with `pip3` is `adafruit-circuitpython-lis3dh`. This the name used on pypi.org ().

Libraries often depend on other libraries. When you install a library with `pip3`, it will automatically install other needed libraries.

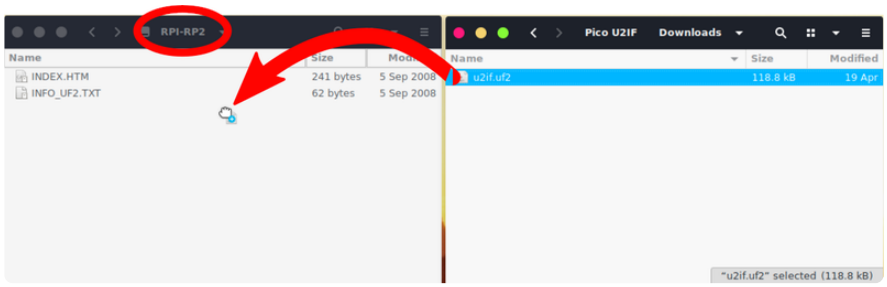
Desktop Computers using a USB Adapter

When you use a desktop computer with a USB adapter, like the MCP2221A, FT232H, or u2if firmware on an RP2040, you will also use pip3. However, do not install the library with `sudo pip3`, as mentioned in some guides. Instead, just install with `pip3`.

MicroPython

For MicroPython, you will not use `pip3`. Instead you can get the library from the CircuitPython bundles. See this guide page () for more information about the bundles, and also see the Libraries page on circuitPython.org ().

Setup for Pico



The first step is to install the u2if firmware () onto the Raspberry Pi Pico. This is super easy:

1. Download the latest release UF2 file from the repo: https://github.com/execuc/u2if/releases ()
2. Put the Pico in bootloader mode by holding the BOOTSEL button while plugging in the board.
3. Drag the downloaded UF2 file to the RPI-RP2 folder.
4. DONE!

The board will reset after the copy is complete. Note that no folders will show up. So it may seem like nothing happened.

No folders will show up after reset - this is normal.

Now the Pico will show up as two devices - a USB HID (Human Interface Device) and a USB CDC (Communication Device Class). The former provides a generic interface for sending 64 byte "reports" back and forth. The later is essentially a serial interface, aka "com port". How these show up on your PC will depend on OS.

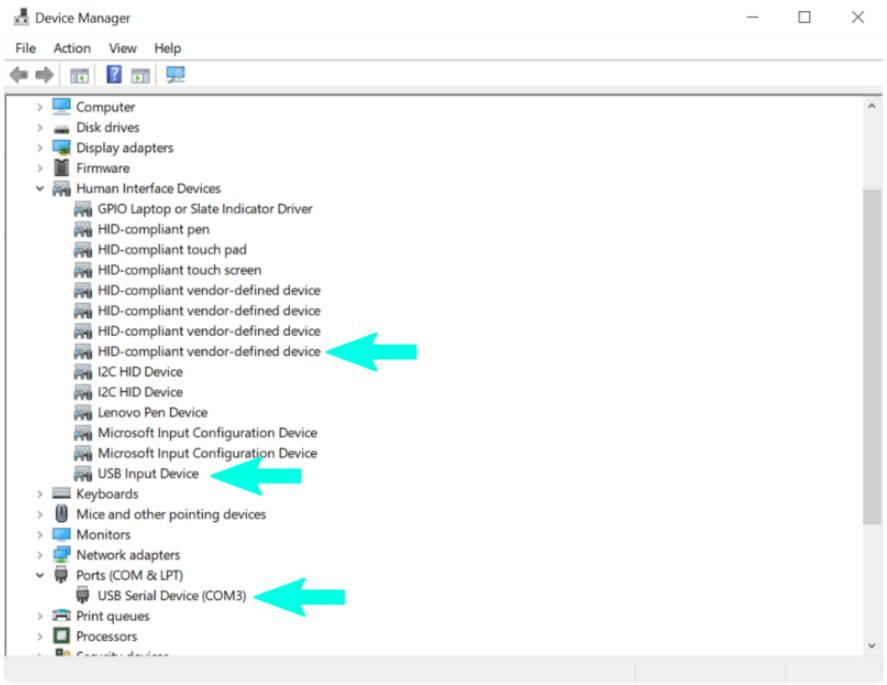
linux dmesg

On linux, the dmesg output will look something like this when connecting the Pico:

```
[Mon Apr 26 13:07:36 2021] usb 2-1.5: new full-speed USB device number 12 using
ehci-pci
[Mon Apr 26 13:07:36 2021] usb 2-1.5: New USB device found, idVendor=cafe,
idProduct=4005, bcdDevice= 1.00
[Mon Apr 26 13:07:36 2021] usb 2-1.5: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
[Mon Apr 26 13:07:36 2021] usb 2-1.5: Product: U2IF
[Mon Apr 26 13:07:36 2021] usb 2-1.5: Manufacturer: Pico
[Mon Apr 26 13:07:36 2021] usb 2-1.5: SerialNumber: 0xE6604430433F5326
[Mon Apr 26 13:07:36 2021] cdc_acm 2-1.5:1.0: ttyACM0: USB ACM device
[Mon Apr 26 13:07:36 2021] hid-generic 0003:CAFE:4005.000C: hiddev1,hidraw7: USB
HID v1.11 Device [Pico U2IF] on usb-0000:00:1d.0-1.5/input2
```

Windows Device Manager

On Windows, several new entries should show up in Device Manager:



Setup on PC

Do NOT pip install hid. That is a different library that should NOT be installed.

The main support for the Pico running the u2if firmware in Blinka utilizes the hidapi library (). Some of the features rely on sending data via a serial connection. For that, we use the pyserial library (). And to allow use of CircuitPython Libraries, we need the Blinka () interface layer.

All of these in turn rely on a several other things which vary for different OS's. So before we can actually use the Pico, we need to get everything setup. See the OS specific sections for what we went through to get things working for each.

Additional Information

Just for reference, here are links to more information about the main Python libraries being used. Here's the README from the hidapi source code repo, which has some install information:

hidapi README

Here's the main documentation for pySerial:

pySerial documentation

The pySerial source code repo is here ().

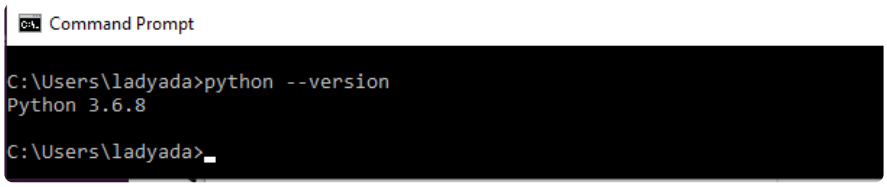
But first try the install instructions on the pages that follow for your OS.

Windows

Have Python 3 Installed

We assume you already have Python 3 installed on your computer. Note we do not support Python 2 - it's deprecated and no longer supported!

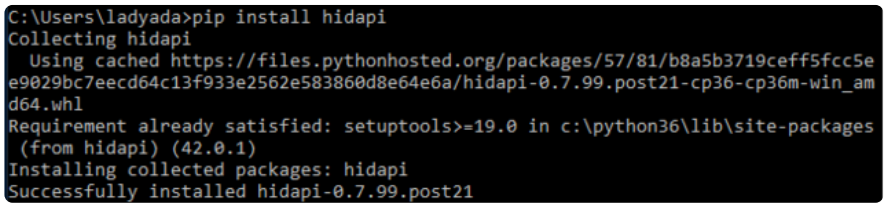
At your command line prompt of choice, check your Python version with `python --version`



Install hidapi

From the command line, manually install hidapi with

```
pip3 install hidapi
```



Do NOT pip install hid. That is a different library that should NOT be installed.

If the install fails with text that ends with something like:

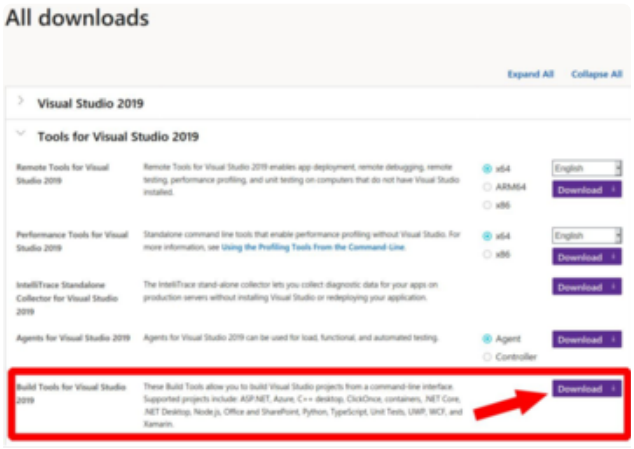
> distutils.errors.DistutilsError: Setup script exited with error: Microsoft Visual C++ 14.0 is required. Get it with "Microsoft Visual C++ Build Tools": https://visualstudio.microsoft.com/downloads/ ()

then you will need to also install the Microsoft Visual C++ Build Tools. Thanks to @jklem for pointing this out in the forums ().

Download it from here (same link as in text):

Microsoft Visual Studio Downloads

NOTE: You do not need the full Visual Studio IDE. Just the Build Tools.



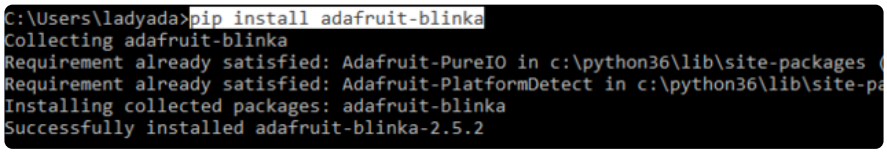
Scroll down to where it says Tools for Visual Studio 2019.
Expand the list to show the sub options.
Click the Download button for Build Tools for Visual Studio 2019.

This downloads a .exe file with a name like vs_BuildTools.exe. Run that to install the build tools and then try the pip install again.

Install Blinka

To install Blinka and its dependencies, run:

```
pip3 install adafruit-blinka
```



Set Environment Variable

You must do this every time before running circuitpython code, you can set it permanently in windows if you like, for now just type into the same cmd window you're using with Python

```
set BLINKA_U2IF=1
```

If you are using Windows Powershell, the syntax is a little different. In that case do:

```
$env:BLINKA_U2IF=1
```

Run the sanity checks.

Now move on to the Post Install Checks section and run the commands there to make sure everything is installed correctly.

Mac OSX

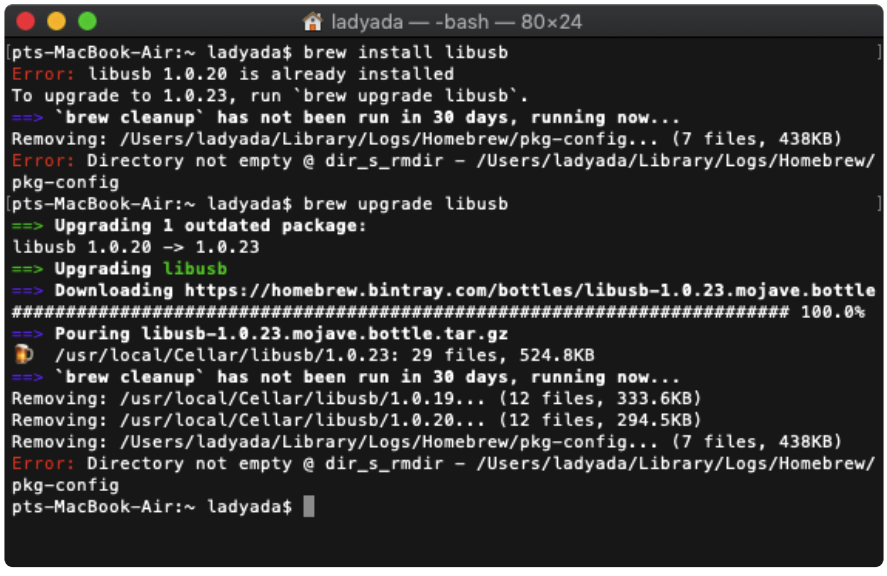
We assume you already have Python 3 and brew available on your Mac. Thankfully, setup on MacOS X is not so bad!

Note: If you are running VMWare Fusion on MacOS, then you can also try the Windows install () process.

Install libusb

Start by installing libusb with

```
brew install libusb
```



Install PySerial

Type `pip3 install pyserial`

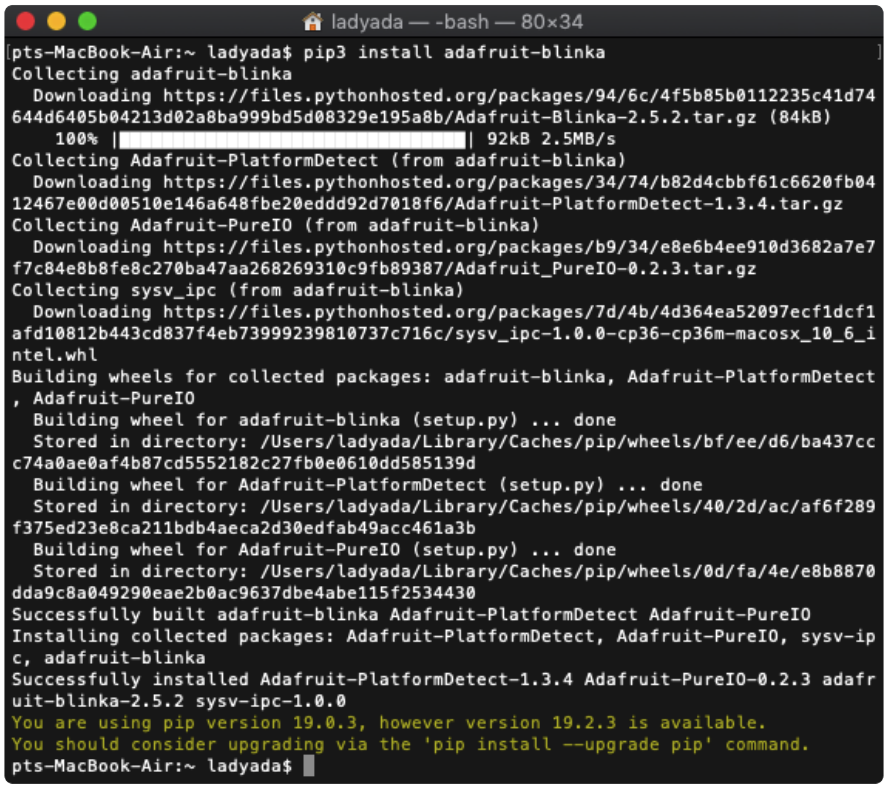
Install hidapi

Type `pip3 install hidapi`

Do NOT pip install hid. That is a different library that should NOT be installed.

Install Blinka

Then `pip3 install adafruit-blinka`



Set Environment Variable

You'll need to set this variable every time before running CircuitPython code. To do this, we set the environment variable `BLINKA_U2IF`.

You can set the variable by running:

```
export BLINKA_U2IF="1"
```

Run the sanity checks.

Now move on to the Post Install Checks section and run the commands there to make sure everything is installed correctly.

Linux

The following shows a typical run through installing and setting things up on Linux.

Install libusb and libudev

Run the following:

```
sudo apt-get install libusb-1.0 libudev-dev
```

and answer Y to the prompt. This should install libusb and libudev.

Setup udev rules

For this, we just follow recommended setup () from the firmware. Use a text editor to create and edit a file named /etc/udev/rules.d/55-u2if.rules and add the following contents:

```
SUBSYSTEM=="usb", ATTR{idVendor}=="cafe", ATTR{idProduct}=="4005", MODE="0666"
```

Install hidapi

To install hidapi, run:

```
pip3 install hidapi
```

Do NOT pip install hid. That is a different library that should NOT be installed.

Install pySerial

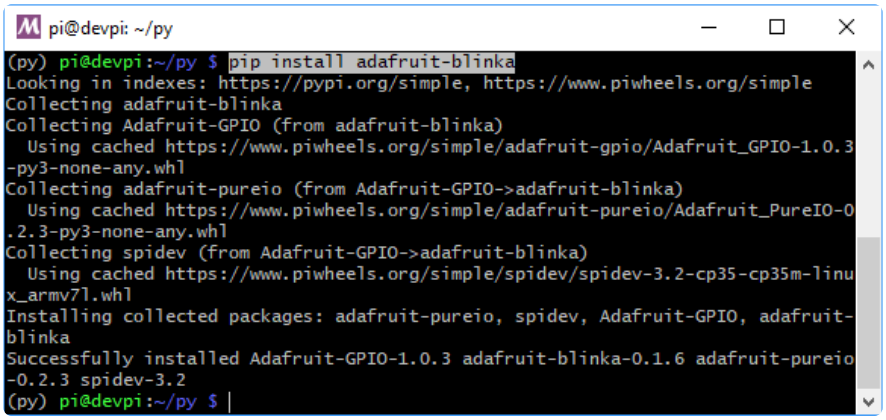
To install pySerial, run:

```
pip3 install pyserial
```

Install Blinka

To install Blinka and its dependencies, run:

```
pip3 install adafruit-blinka
```



Set environment variable

We need to manually signal to Blinka that we have a Pico running the u2if firmware. To do this we set the environment variable BLINKA_U2IF. The value doesn't matter, just use 1:

```
export BLINKA_U2IF=1
```

Don't forget this step. Things won't work unless BLINKA_U2IF is set.

Run the sanity checks.

Now move on to the Post Install Checks section and run the commands there to make sure everything is installed correctly.

Post Install Checks

After going through all the install steps for your OS, run these checks as simple tests to make sure everything is installed correctly. Go ahead and plug in your Pico to a USB port on your PC.

Most of these tests are done via the Python REPL, at the `>>>` prompt. To get there, simply launch Python:

```
$ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
&gt;&gt;&gt;
```

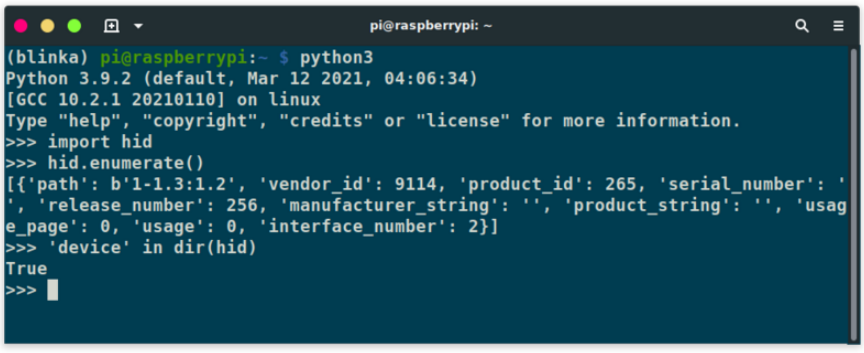
Make sure you've set the BLINKA_U2IF environment variable.

Check that hidapi is installed correctly

At the Python REPL, type:

```
import hid
hid.enumerate()
'device' in dir(hid)
```

The `enumerate()` command should dump a listing of everything attached to your USB ports. The last command is a test of the actual hid module imported and should return `True`.



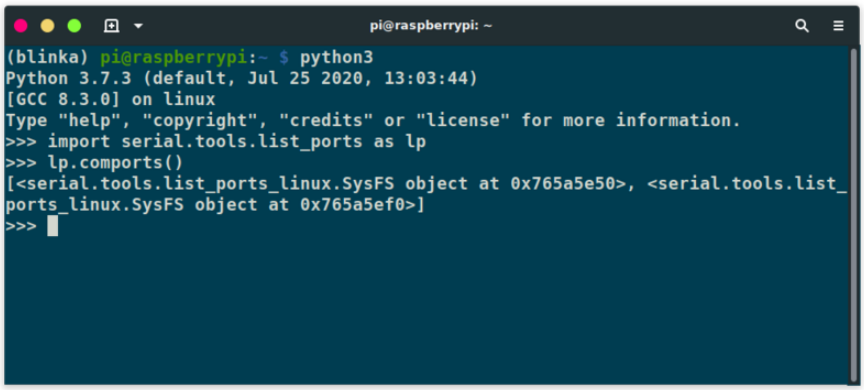
If the 'device' check returned False, make sure the library installed is hidapi. There is a separate library named hid which should NOT be installed.

Check that pySerial is installed correctly

At the Python REPL, type:

```
import serial.tools.list_ports as lp
lp.comports()
```

You should get a list of available COM ports.



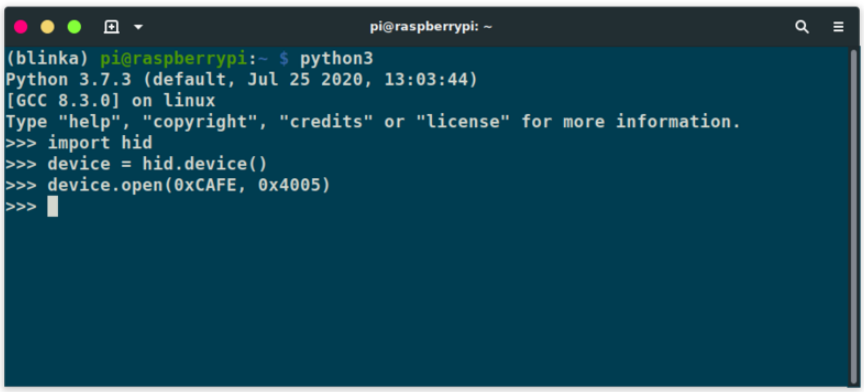
Check that Pico can be found

At the Python REPL, type:

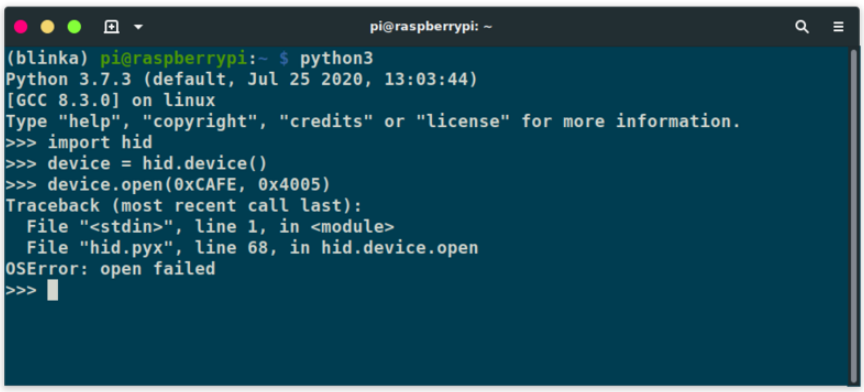
```
import hid
device = hid.device()
device.open(0xCAFE, 0x4005)
```

See the "Other RP2040 Boards" section for USB VID and PID to use with open() for non-Pico boards.

It should run without any errors:



If for some reason the Pico can not be found, you might see something like this:



Check your USB cable connection and double check that the u2if firmware is loaded.

If you want to continue testing in the same Python session, then make a quick call to `close()` to free up the device.

```
device.close()
```

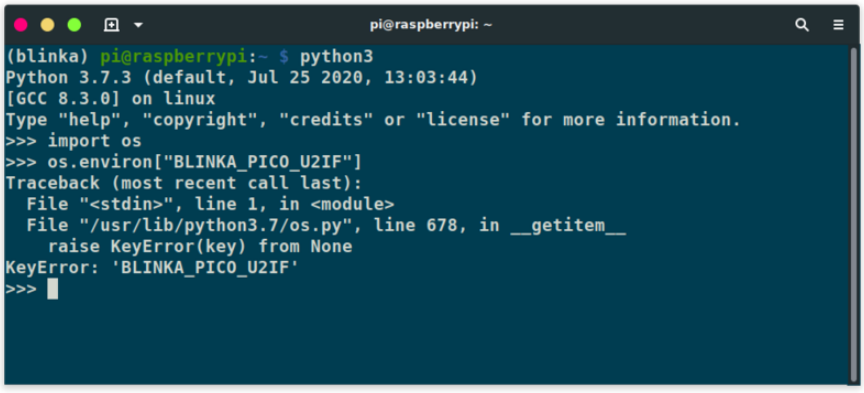
Or, just exit the Python session.

Check environment variable within Python

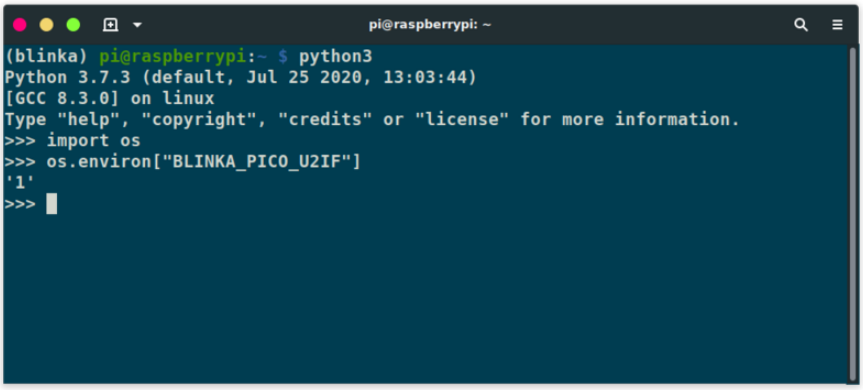
At the Python REPL, type:

```
import os
os.environ["BLINKA_U2IF"]
```

If you get a KeyError it means you did not set the environment variable right:



If you have set it correctly, you'll get a value back:

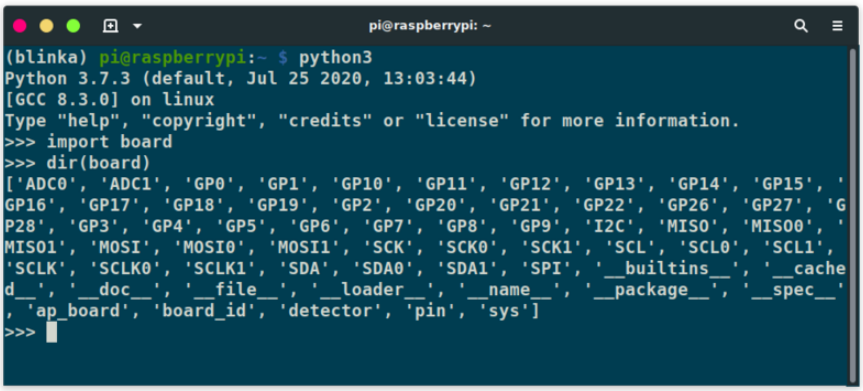


Check Blinka is setup correctly

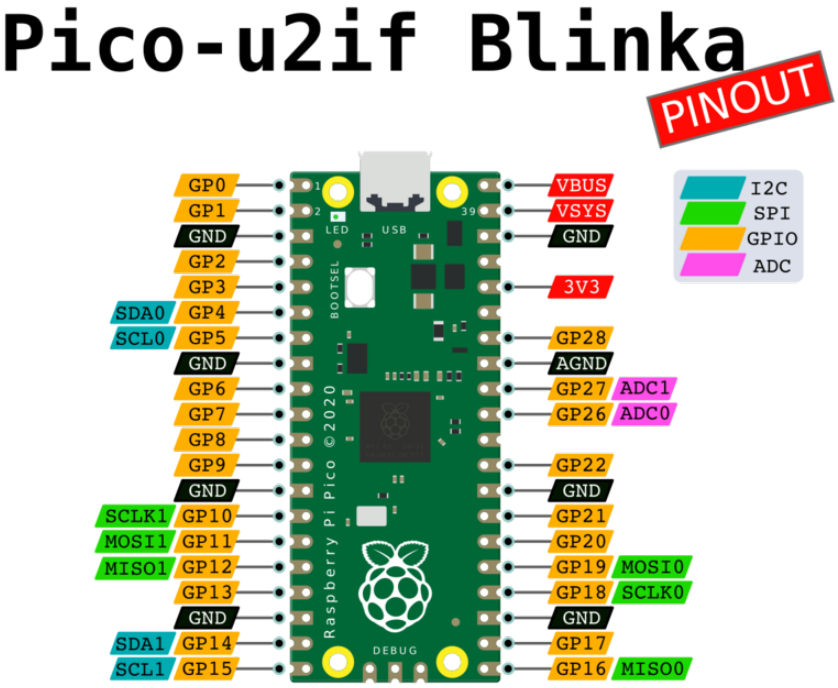
If all of the above checks pass, go ahead and try this as a quick sanity check that basic Blinka functionality is in place. At the Python REPL, type:

```
import board
dir(board)
```

You should not get any errors and the various pins available on the Pico should be shown.



Pinout



While the Raspberry Pi Pico allows the I2C and SPI pins to appear in multiple locations, the u2if firmware fixes these locations to specific pins.

Power Pins

- VBUS - micro-USB input voltage
- VSYS - main system input voltage
- 3V3 - regulated 3.3V output, 300mA max
- GND - main ground reference
- AGND - ground reference for GP26-29 and ADC0 and ADC1

GPIO Pins

- GP0 to GP28 - General Purpose Input Output (GPIO) as well as Pulse Width Modulation (PWM)

I2C Pins

- SCL0 - I2C port 0 clock
- SDA0 - I2C port 0 data
- SCL1 - I2C port 1 clock

- SDA1 - IC2 port 1 data

SPI Pins

- SCLK0 - SPI port 0 clock
- MOSI0 - SPI port 0 data out
- MISO0 - SPI port 0 data in
- SCLK1 - SPI port 1 clock
- MOSI1 - SPI port 1 data out
- MISO1 - SPI port 1 data in

ADC Pins

- ADC0 - Analog to Digital Converter (ADC) 0
- ADC1 - Analog to Digital Converter (ADC) 1

You are correct in noting that ADC2 is not exposed, we are not sure why!

Examples

All right, now that all that annoying install stuff is done, let's have some fun.

The following sections will provide some basic examples for the main use cases - GPIO, ADC, PWM, I2C, SPI, and NeoPixel.

Make sure you've set the BLINKA_U2IF environment variable.

Installing Libraries for Breakouts

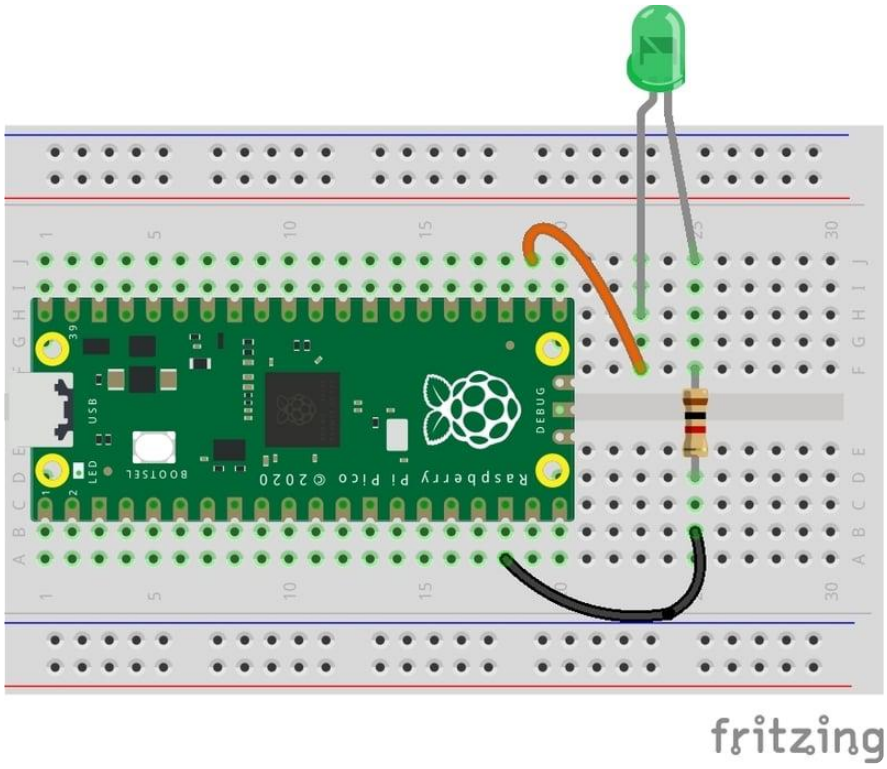
The general process for installing the CircuitPython library you are interested in will be the same as shown in the Python section of the Learn guide for your sensor. Just use pip3.

GPIO

Digital Output

Let's blink a LED!

Here's the bread board layout. The resistor can be something around 1kOhm. We don't need to make the LED super bright.



And here's a complete blink program you can run to make the LED blink forever.

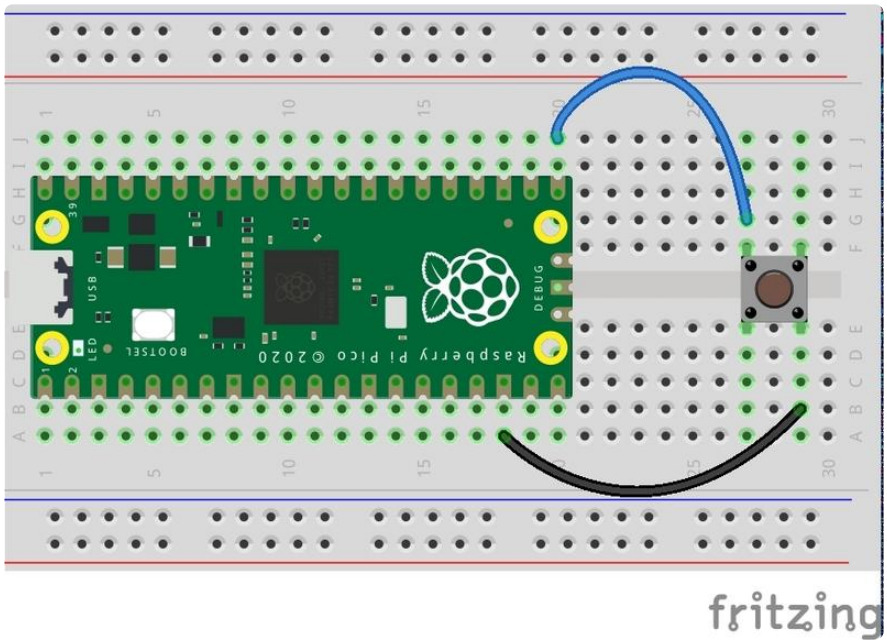
```
import time
import board
import digitalio

led = digitalio.DigitalInOut(board.GP17)
led.direction = digitalio.Direction.OUTPUT

while True:
    led.value = True
    time.sleep(0.5)
    led.value = False
    time.sleep(0.5)
```

Digital Input

Let's read a button!

The cool thing here is that the Pico has internal pull up resistors. Therefore we don't need to add any additional external resistors, which you might see in some other wiring diagrams. The equivalent resistor is inside the Pico!

Here's the breadboard layout.



Here's the code to run. It will continuously print the button state.

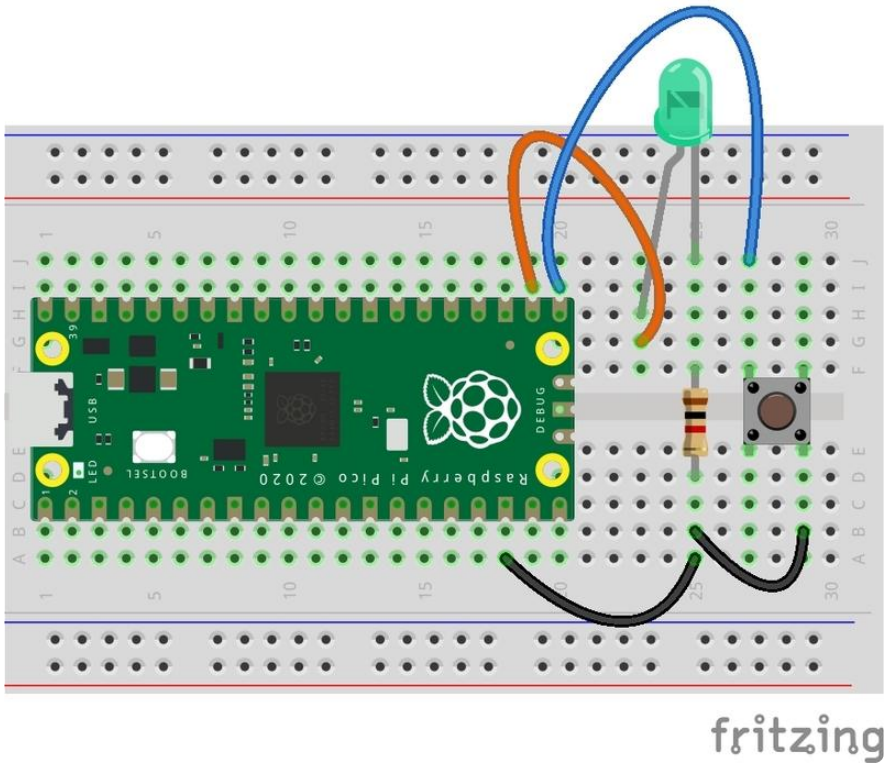
- `True` = not pressed
- `False` = pressed

```
import board
import digitalio

button = digitalio.DigitalInOut(board.GP16)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP

while True:
    print(button.value)
```

Digital Input and Output

Ok, let's put those two together and make the button turn on the LED. So we'll use two digital pins - one will be an input (button) and one will be an output (LED).

Here's the bread board layout.



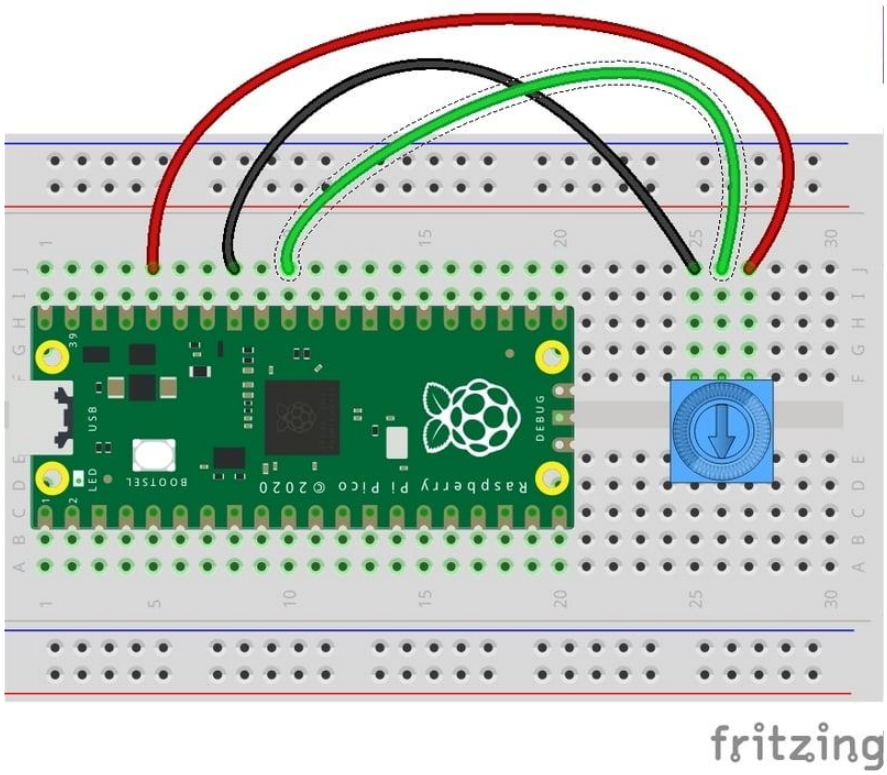
And here's the code. Note how the code uses `not` to invert the button logic.

```
import board
import digitalio

led = digitalio.DigitalInOut(board.GP17)
led.direction = digitalio.Direction.OUTPUT

button = digitalio.DigitalInOut(board.GP16)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP

while True:
    led.value = not button.value
```

ADC

Let's read an analog signal!

For this, we'll use a small 10k trim pot () to set up a voltage divider. Here's the wiring diagram:



And here's the code:

```
import time
import board
import analogio

knob = analogio.AnalogIn(board.ADC0)

def get_voltage(raw):
    return (raw * 3.3) / 65536

while True:
    raw = knob.value
    volts = get_voltage(raw)
    print("raw = {:5d} volts = {:5.2f}".format(raw, volts))
    time.sleep(0.5)
```

Spin the knob and the values should change.

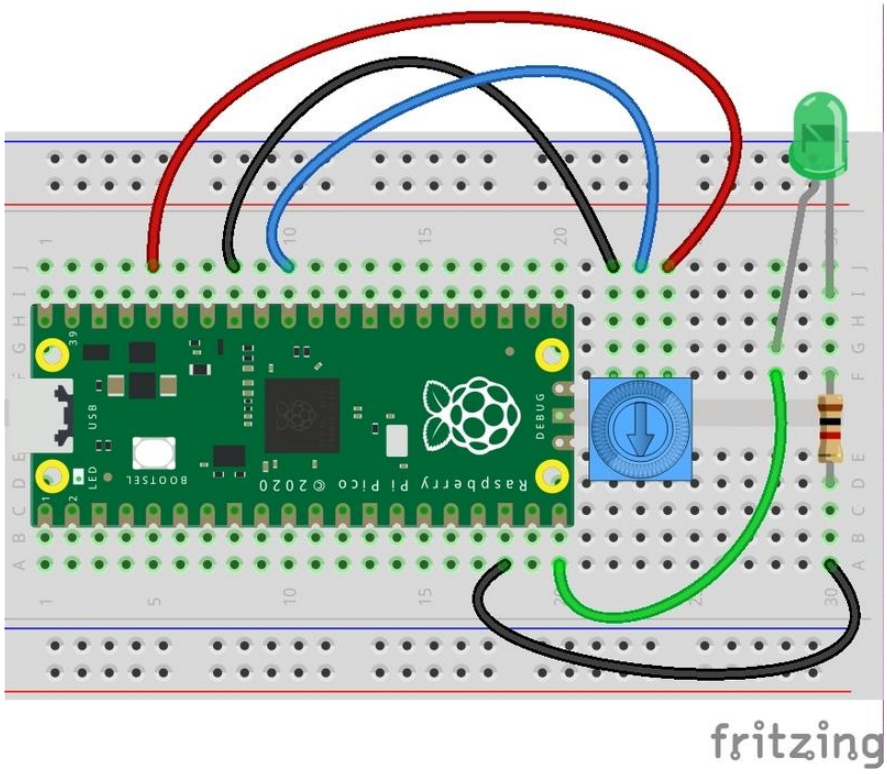
Note that even though the Pico's ADC is 12 bits, the value is scaled to 16 bits to comply with the CircuitPython API.

PWM

Let's dim an LED!

To do this we will use Pulse Width Modulation (PWM) output. The `duty_cycle` of the PWM output will control the LED brightness. We'll combine this with the previous ADC

example so we can use the knob to control the LED brightness. Here's the breadboard layout:



And here's the code to run:

```
import board
import pwmio
import analogio

knob = analogio.AnalogIn(board.ADC0)

led = pwmio.PWMOut(board.GP15, frequency=1000)

while True:
    led.duty_cycle = knob.value
```

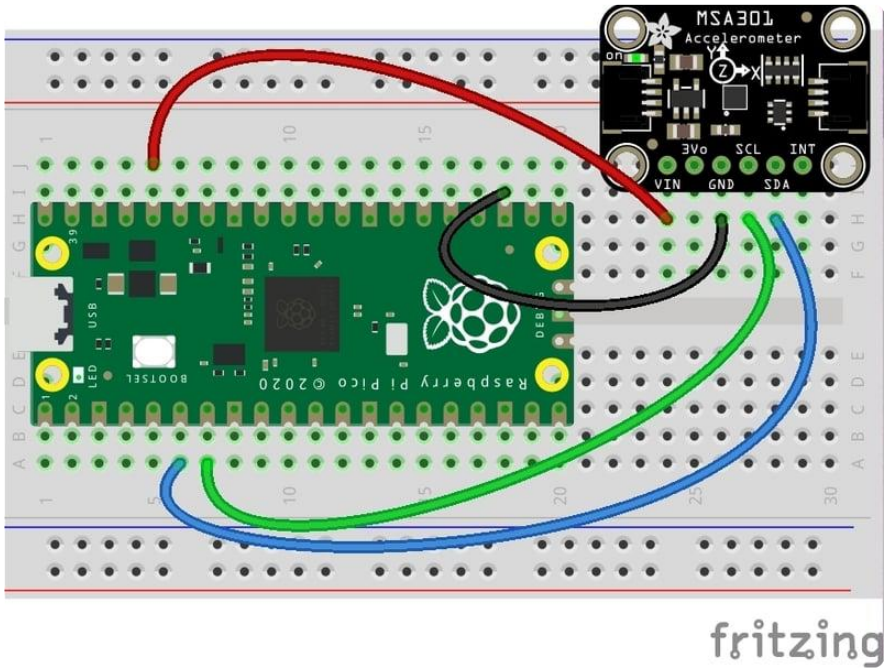
Turn the knob and the LED should get dimmer and brighter.

I2C

Let's talk to an I2C sensor!

The Pico has two I2C ports. Remember that you can attach multiple sensors to a single port as long as each has a unique I2C address. So you don't need to use two just because you have two sensors.

We'll use the MSA301 sensor () which can read acceleration. Here we show wiring via the header pins. But if you wanted to use the STEMMA QT connector, you could by using one of the pigtail breakout cables.



Trying to use an I2C port with nothing attached can cause the system to hang.

I2C0 is the default port used by board.I2C() and SCL/SDA pins.

Install MSA301 Library

To install the MSA301 library, run the following:

```
sudo pip3 install adafruit-circuitpython-msa301
```

Note that this step is the same as shown in the main MSA301 guide (). You would do the same general process for any other sensor with a CircuitPython library.

Example Code

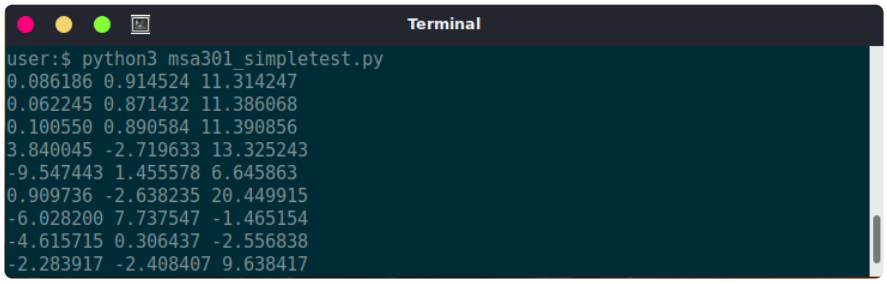
And then we can run the example from the library. Download it from here:

MSA301 Simple Test Example

save it as msa301_simpletest.py and run it with:

```
python3 msa301_simpletest.py
```

Pick up the board and spin it around. You should see the values change:



Live Plot Example

This one is a little fancier and requires matplotlib () to be installed on the host PC as well. This is the example shown running in the guide thumbnail image.

Here's the code:

```
import board
import busio
import adafruit_msa301

import matplotlib.pyplot as plt
import matplotlib.animation as animation
from collections import deque
import time

i2c = busio.I2C(board.SCL1, board.SDA1)

msa = adafruit_msa301.MSA301(i2c)

REFRESH_RATE = 50
HIST_SIZE = 61

x_time = [x * REFRESH_RATE for x in range(HIST_SIZE)]
x_time.reverse()

y_data = [deque([None] * HIST_SIZE, maxlen=HIST_SIZE) for _ in range(3)]

fig, ax = plt.subplots(1, 1)
fig.canvas.manager.set_window_title("MSA301 Acceleration")
fig.set_figwidth(9)
fig.set_figheight(3)

ax.grid(True, linestyle=':')
ax.set_facecolor('#303030')
ax.set_xlim(min(x_time), max(x_time))
ax.set_ylim(-15, 15)
ax.invert_xaxis()

lines = []
for data in y_data:
    line, = ax.plot(x_time, data)
    lines.append(line)

lines[0].set_color('#d1ff7a'); lines[0].set_linewidth(3)
lines[1].set_color('#7af6ff'); lines[1].set_linewidth(3)
```

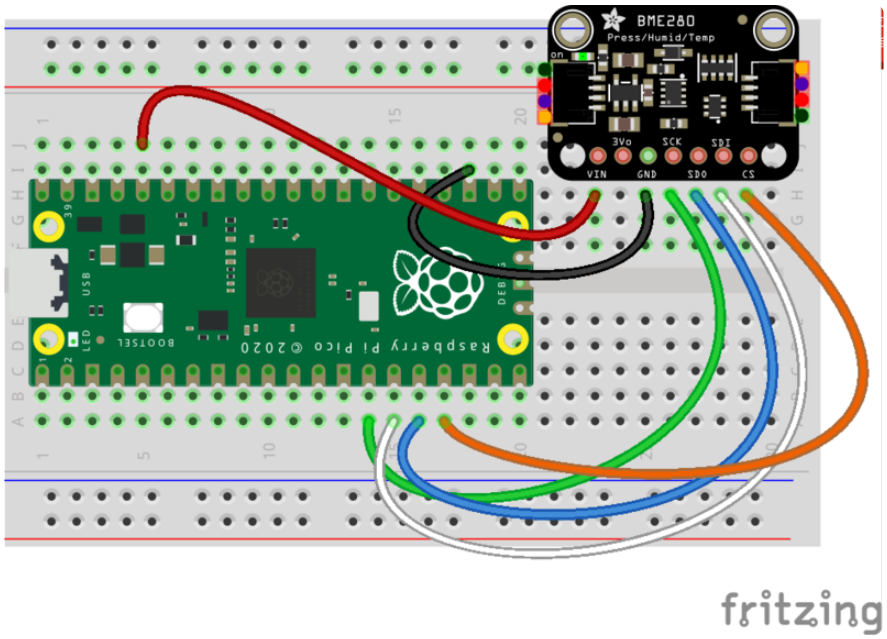
```
lines[2].set_color('#ff36fc'); lines[2].set_linewidth(3)

def animate(foo):
    for i, a in enumerate(msa.acceleration):
        y_data[i].append(a)
        lines[i].set_ydata(y_data[i])
    fig.canvas.draw()

ani = animation.FuncAnimation(fig, animate, interval=REFRESH_RATE)
plt.show()
```

SPI

Let's talk to a SPI sensor.

The Pico has two SPI ports. Remember that you can attach multiple sensors to a single port as long as each has a separate chip select (CS) pin.

Here we use a BME280 sensor on the secondary SPI port.



SPI0 is the default port used by board.SPI() and MOSI/MISO/SCLK pins.

Install the BME280 Library

To install the BME280 library, run the following:

```
sudo pip3 install adafruit-circuitpython-bme280
```

Note that this step is the same as shown in the main BME280 guide (). You would do the same thing for any other sensor.

Run Example

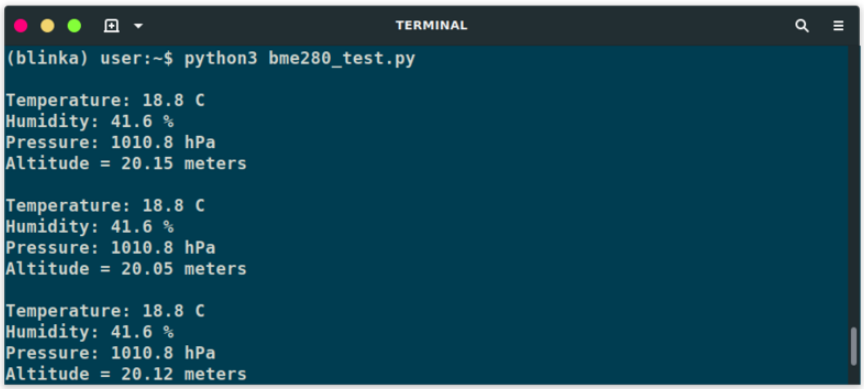
Here's is the example code to run:

```
import time
import board
import busio
import digitalio
import adafruit_bme280

spi = busio.SPI(board.SCK1, board.MOSI1, board.MISO1)

cs = digitalio.DigitalInOut(board.GP13)

bme280 = adafruit_bme280.Adafruit_BME280_SPI(spi, cs)

while True:
    print("\nTemperature: %0.1f C" % bme280.temperature)
    print("Humidity: %0.1f %%" % bme280.relative_humidity)
    print("Pressure: %0.1f hPa" % bme280.pressure)
    print("Altitude = %0.2f meters" % bme280.altitude)
    time.sleep(2)
```

Save this as something like bme280_test.py and run it with:

```
python3 bme280_test.py
```

and you should see it print out sensor readings over and over:

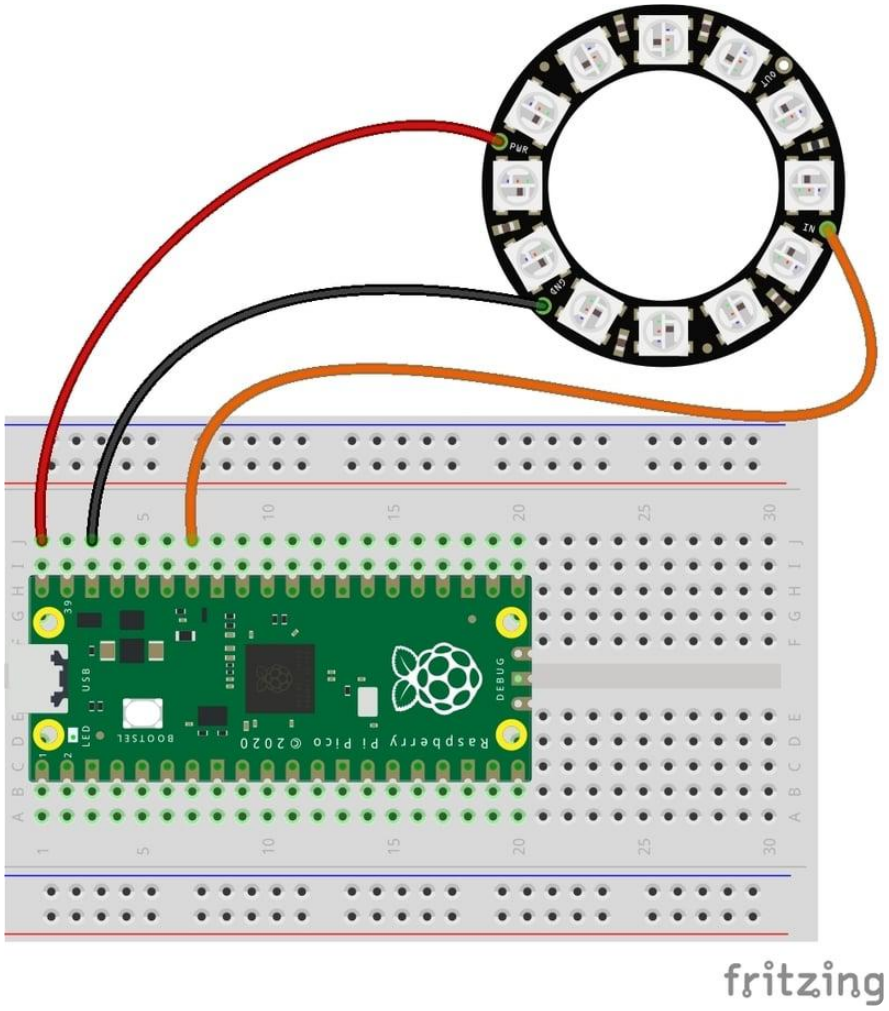


NeoPixel

Let's light up some NeoPixels!

We could use the SPI port to do this, using the hack provided by the neopixel_spi library (). But the u2if firmware supports the real deal. No need for the hack. Just wire NeoPixels to any available GP pin and use the normal neopixel library.

Here's an example wiring:



This example uses a 12 ring RGB NeoPixel (). For any other setup, just change the number of pixels and possibly the pixel order.

Currently, only RGB NeoPixels are supported.

While NeoPixels are best used with 5V power and 5V logic, many times they are fine with 3.3V logic. If you don't get the LEDs to light up, try powering the ring from 3.3V or adding a level shifter

Install NeoPixel Library

To install the NeoPixel library, run the following:

```
sudo pip3 install adafruit-circuitpython-neopixel
```

These are the same install instructions as found in the main NeoPixel guide ().

Run Example

And here is the example code to drive the 12 NeoPixel ring. To keep things simple, we simply fill the ring with various colors.

```
import time
import board
import neopixel

COLORS = (
    (255,   0,   0),
    (  0, 255,   0),
    (  0,   0, 255),
    (255, 255,   0),
    (255,   0, 255),
    (  0, 255, 255),
)

pixels = neopixel.NeoPixel(board.GP28, 12)

while True:
    for color in COLORS:
        pixels.fill(color)
        time.sleep(1)
```

Save that as something like neopixel_ring.py and then run with the following:

```
python3 neopixel_ring.py
```

And the ring should light up!

Other RP2040 Boards

Since the u2if firmware uses standard HID and CDC interfaces for communicating with the host PC, it can potentially run on any Raspberry Pi RP2040 based board, not just the Pico. The main code changes needed are:

- Provide appropriate USB PID and VID.
- Change pin mappings to specific RP2040 based board.

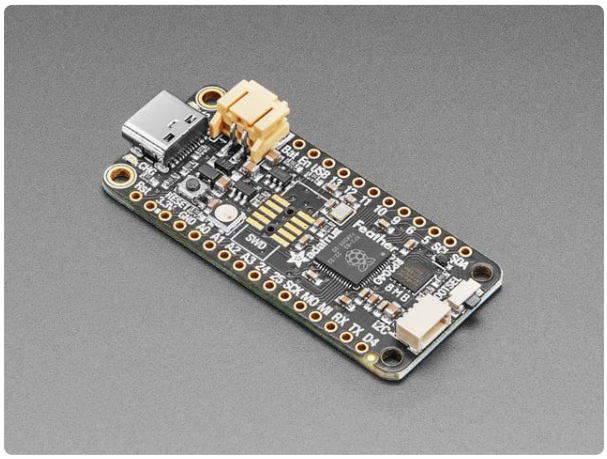
We've done that for several Adafruit RP2040 based boards. Details for each are provided below. For each, install the provided UF2 firmware, set the environment variable:

```
BLINKA_U2IF=1
```

and then launch Python. The board will be auto detected based on USB PID and VID.

Make sure you've also updated to the latest versions of Adafruit Blinka and PlatformDetect.

Feather RP2040



Adafruit Feather RP2040
A new chip means a new Feather, and the Raspberry Pi RP2040 is no exception. When we saw this chip we thought "this chip is going to be awesome when we give it the Feather...
https://www.adafruit.com/product/4884

Here is the firmware:

u2if_feather_rp2040.uf2

Pico Firmware USB IDs:

- `USB_VID = 0x239A`
- `USB_PID = 0x00F1`

Example check-if-found test code:

```
import hid
device = hid.device()
device.open(0x239A, 0x00F1)
```

Here is what you should see if you list the `board` pins:

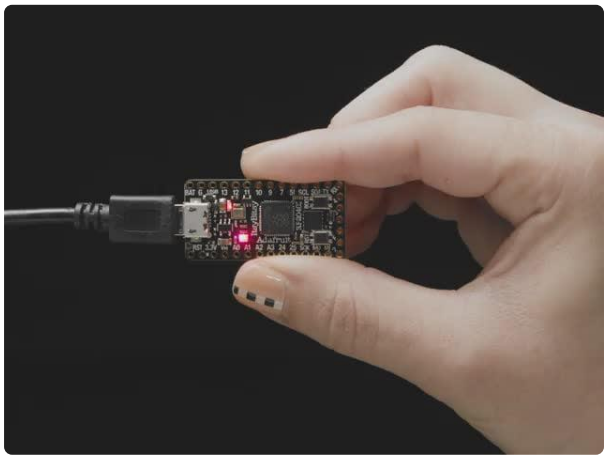
```
$ python3
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
&gt;&gt;&gt; import board
&gt;&gt;&gt; dir(board)
['A0', 'A1', 'A2', 'D0', 'D1', 'D10', 'D11', 'D12', 'D13', 'D24', 'D25', 'D4', 'D5',
```

```
'D6', 'D9', 'I2C', 'MISO', 'MOSI', 'SCK', 'SCL', 'SCLK', 'SDA', 'SPI',
'__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'ap_board', 'board_id', 'detector', 'pin', 'sys']
&gt;&gt;&gt;
```

Here is an example that scans for connected I2C devices. Make sure something is actually connected to the SCL/SDA pins or the STEMMA QT connector.

```
import board

i2c = board.I2C()

i2c.try_lock()
i2c.scan()
i2c.unlock()
```

ItsyBitsy RP2040



Adafruit ItsyBitsy RP2040
A new chip means a new ItsyBitsy, and the Raspberry Pi RP2040 is no exception. When we saw this chip we thought "this chip is going to be awesome when we give it the ItsyBitsy...
https://www.adafruit.com/product/4888

Here is the firmware:

u2if_itsybitsy_rp2040.uf2

Pico Firmware USB IDs:

- `USB_VID = 0x239A`
- `USB_PID = 0x00FD`

Example check-if-found test code:

```
import hid
device = hid.device()
device.open(0x239A, 0x00FD)
```

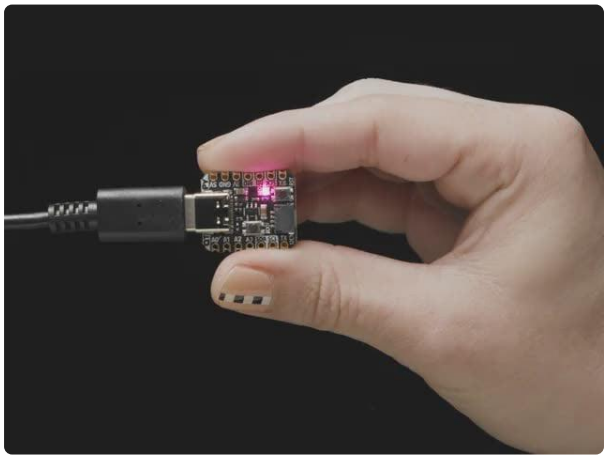
Here is what you should see if you list the `board` pins:

```
$ python3
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
&gt;&gt;&gt; import board
&gt;&gt;&gt; dir(board)
['A0', 'A1', 'A2', 'BUTTON', 'D0', 'D1', 'D10', 'D11', 'D12', 'D13', 'D2', 'D24',
'D25', 'D3', 'D4', 'D5', 'D7', 'D9', 'I2C', 'MISO', 'MOSI', 'NEOPIXEL',
'NEOPIXEL_POWER', 'SCK', 'SCL', 'SCLK', 'SDA', 'SPI', '__builtins__', '__cached__',
'__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__',
'ap_board', 'board_id', 'detector', 'pin', 'sys']
&gt;&gt;&gt;
```

Here is a simple example program that reads the state of the BOOT button.

```
import time
import board
import digitalio

button = digitalio.DigitalInOut(board.BUTTON)
button.direction = digitalio.Direction.INPUT

while True:
    # value is False when button is pressed
    if not button.value:
        print("Button pressed!")
        time.sleep(0.1)
```

QT Py RP2040



Adafruit QT Py RP2040

What a cutie pie! Or is it... a QT Py? This diminutive dev board comes with one of our new favorite chip, the RP2040. It's been made famous in the new https://www.adafruit.com/product/4900

Here is the firmware:

u2if_qtpy_rp2040.uf2

Pico Firmware USB IDs:

- `USB_VID = 0x239A`
- `USB_PID = 0x00F7`

Example check-if-found test code:

```
import hid
device = hid.device()
device.open(0x239A, 0x00F7)
```

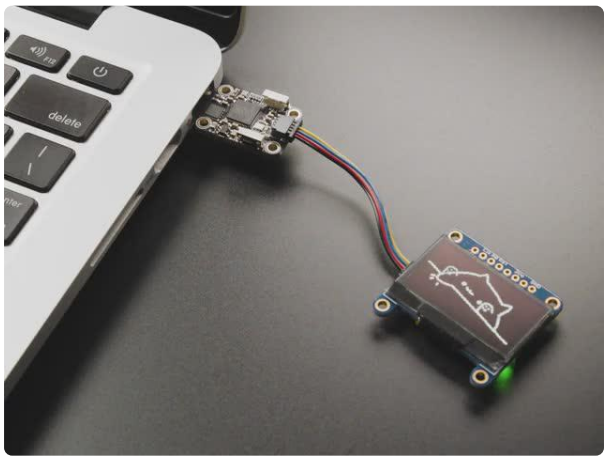
Here is what you should see if you list the `board` pins:

```
$ python3
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
&gt;&gt;&gt; import board
&gt;&gt;&gt; dir(board)
['A1', 'A2', 'A3', 'BUTTON', 'D0', 'D1', 'D10', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7',
'D8', 'D9', 'I2C', 'MISO', 'MOSI', 'NEOPIXEL', 'NEOPIXEL_POWER', 'SCK', 'SCL',
'SCL1', 'SCLK', 'SDA', 'SDA1', 'SPI', '__builtins__', '__cached__', '__doc__',
'__file__', '__loader__', '__name__', '__package__', '__spec__', 'ap_board',
'board_id', 'detector', 'pin', 'sys']
&gt;&gt;&gt;
```

And here is a simple example to light the onboard NeoPixel:

```
import board
import digitalio
import neopixel

pixel = neopixel.NeoPixel(board.NEOPIXEL, 1)

neopwr = digitalio.DigitalInOut(board.NEOPIXEL_POWER)
neopwr.direction = digitalio.Direction.OUTPUT
neopwr.value = True

pixel.fill(0xADAF00)
```

Trinkey QT2040



Adafruit Trinkey QT2040 - RP2040 USB Key with Stemma QT

It's half USB Key, half Adafruit QT Py, and a lotta RP2040...it's Trinkey QT2040, the circuit board with an RP2040 heart and Stemma QT legs....

https://www.adafruit.com/product/5056

Here is the firmware:

u2if_trinkey_qt2040_rp2040.uf2

Pico Firmware USB IDs:

- `USB_VID = 0x239A`
- `USB_PID = 0x0109`

Example check-if-found test code:

```
import hid
device = hid.device()
device.open(0x239A, 0x0109)
```

Here is what you should see if you list the `board` pins:

```
$ python3
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import board
>>> dir(board)
['BUTTON', 'I2C', 'NEOPIXEL', 'SCL', 'SDA', '__builtins__', '__cached__',
'__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__',
'ap_board', 'board_id', 'detector', 'pin', 'sys']
>>>
```

Here is a simple example program that reads the state of the BOOT button.

```
import time
import board
import digitalio

button = digitalio.DigitalInOut(board.BUTTON)
button.direction = digitalio.Direction.INPUT

while True:
    # value is False when button is pressed
    if not button.value:
        print("Button pressed!")
        time.sleep(0.1)
```

Note: For the Trinkey QT2040: Vendor ID is 0x239A and Product ID is 0x0109