May Pad Macropad with the KB2040, KMK, and CircuitPython

Created by Eva Herrada

https://learn.adafruit.com/maypad-macropad-with-the-kb2040-kmk-and-circuitpython

Last updated on 2024-06-03 03:32:56 PM EDT

Table of Contents

Overview

Image credit: keyhive.xyz (https://adafru.it/Yek)

In this guide, you'll learn how to set up your May Pad macropad to use KMK firmware. You'll use the May Pad, designed by /u/reggatronics (https://adafru.it/Yel), and the new Adafruit KB2040, to build a macropad that you'll put KMK firmware on and will also learn how to remap it and create your own configuration.

What is KMK?

> KMK is a feature-rich and beginner-friendly firmware for computer keyboards written and configured in CircuitPython (https://adafru.it/tB7).

What is the May Pad?

> A through hole kit using a pro micro footprint and through hole diodes! It can be a 20 key macropad or a numpad!
>
> Printed cases here: https://www.thingiverse.com/thing:4116560 (https://adafru.it/11hb)
>
> It includes everything you need except for switches and a USB cable
>
> - PCB
> - bottom plate
> - diodes
> - pro micro

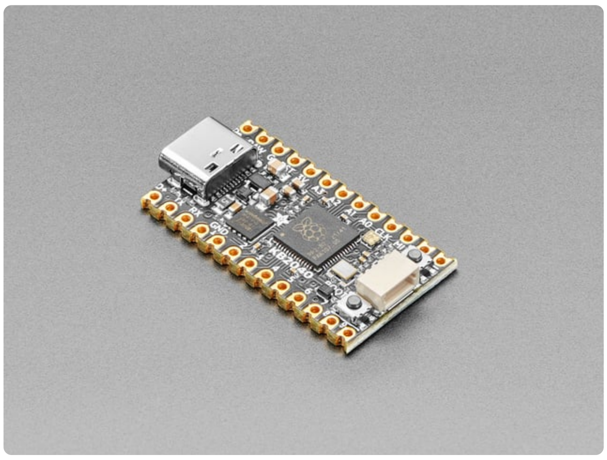
- screws
- standoffs

Parts

May Pad Kit (https://adafru.it/Yek)

A through hole kit using a pro micro footprint and through hole diodes! Can be a 20 key macropad or a numpad!

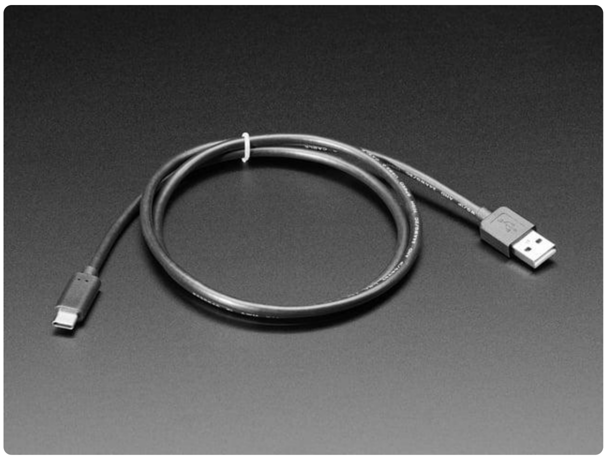
$18.00



Adafruit KB2040 - RP2040 Kee Boar Driver

A wild Kee Boar appears! It's a shiny KB2040! An Arduino Pro Micro-shaped board for Keebs with RP2040. (#keeblife 4 evah) A lot of folks like using Adafruit...

https://www.adafruit.com/product/5302



USB Type A to Type C Cable - approx 1 meter / 3 ft long
As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...
https://www.adafruit.com/product/4474

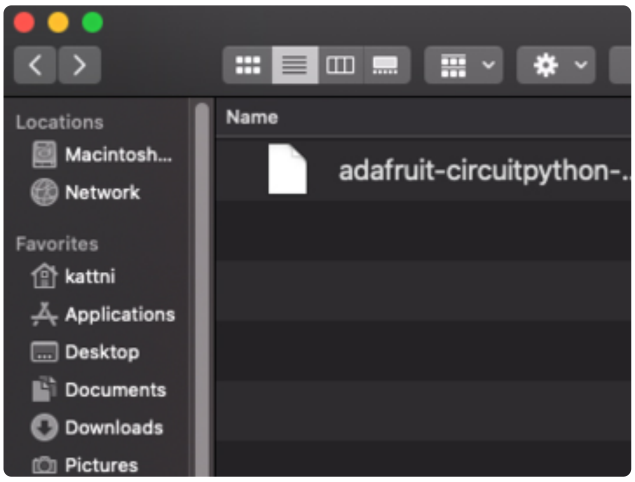
CircuitPython

CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

CircuitPython Quickstart

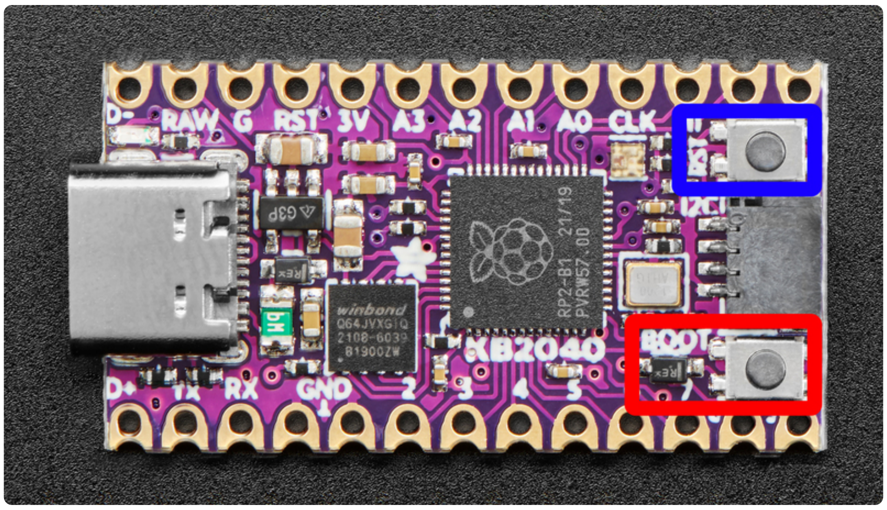
Follow this step-by-step to quickly get CircuitPython running on your board.

Download the latest version of CircuitPython for this board via circuitpython.org
https://adafru.it/Xdr



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.

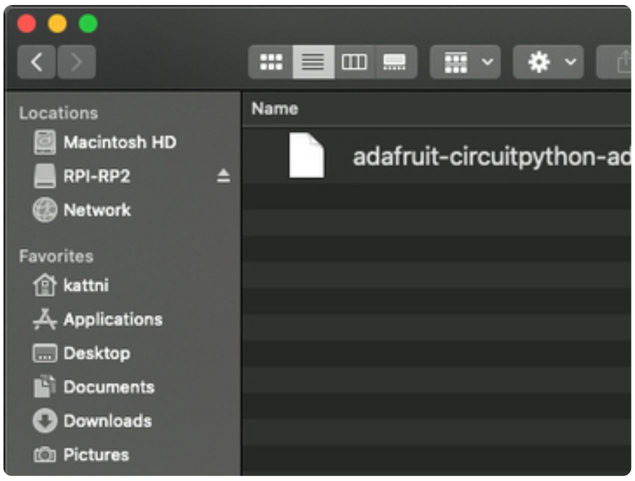


To enter the bootloader, hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset button** (highlighted in blue above). **Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!**

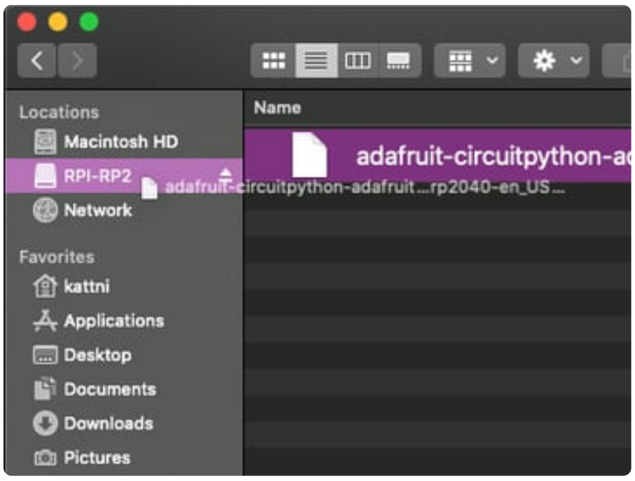
If the drive does not appear, release all the buttons, and then repeat the process above.

You can also start with your board unplugged from USB, press and hold the BOOTSEL button (highlighted in red above), continue to hold it while plugging it into USB, and wait for the drive to appear before releasing the button.

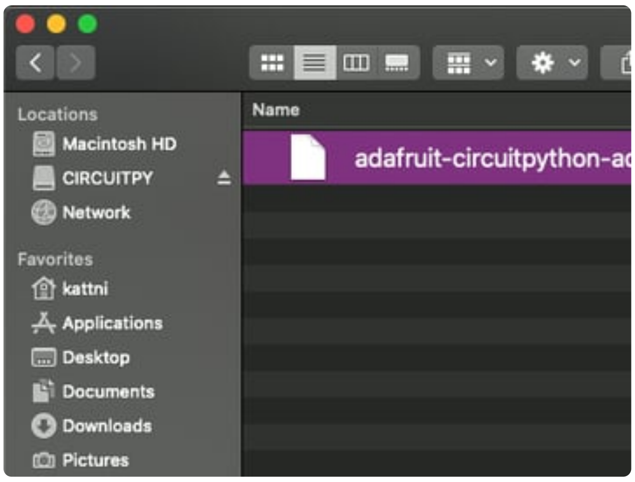
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **RPI-RP2**.



Drag the **adafruit_circuitpython_etc.uf2** file to **RPI-RP2.**



The **RPI-RP2** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it, you're done! :)

Safe Mode

You want to edit your **code.py** or modify the files on your **CIRCUITPY** drive, but find that you can't. Perhaps your board has gotten into a state where **CIRCUITPY** is read-only. You may have turned off the **CIRCUITPY** drive altogether. Whatever the reason, safe mode can help.

Safe mode in CircuitPython does not run any user code on startup, and disables auto-reload. This means a few things. First, safe mode bypasses any code in **boot.py** (where you can set **CIRCUITPY** read-only or turn it off completely). Second, it does not run the code in **code.py**. And finally, it does not automatically soft-reload when data is written to the **CIRCUITPY** drive.

Therefore, whatever you may have done to put your board in a non-interactive state, safe mode gives you the opportunity to correct it without losing all of the data on the **CIRCUITPY** drive.

Entering Safe Mode

To enter safe mode when using CircuitPython, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED (highlighted in green above) will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

In Safe Mode

If you successfully enter safe mode on CircuitPython, the LED will intermittently blink yellow three times.

If you connect to the serial console, you'll find the following message.

```
Auto-reload is off.
Running in safe mode! Not running saved code.

CircuitPython is in safe mode because you pressed the reset button during boot.
Press again to exit safe mode.

Press any key to enter the REPL. Use CTRL-D to reload.
```

You can now edit the contents of the **CIRCUITPY** drive. Remember, your code will not run until you press the reset button, or unplug and plug in your board, to get out of safe mode.

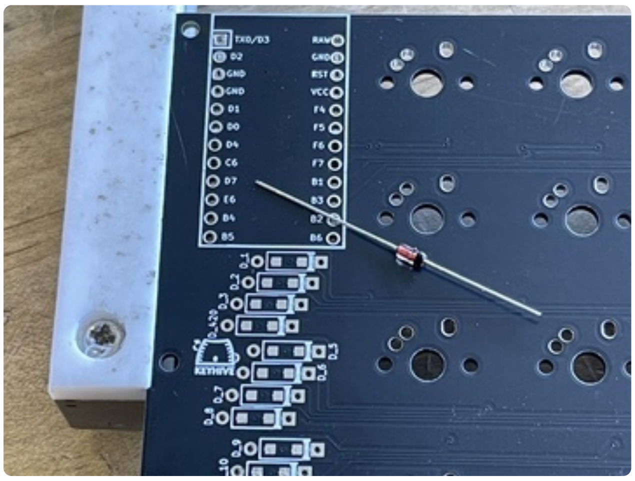
Flash Resetting UF2

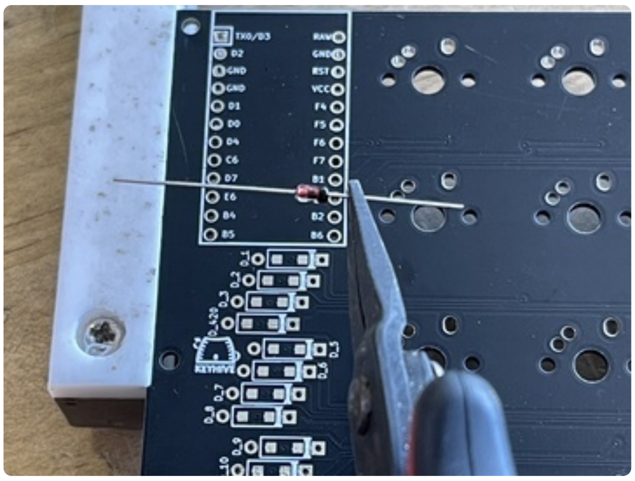
If your board ever gets into a really weird state and CIRCUITPY doesn't show up as a disk drive after installing CircuitPython, try loading this 'nuke' UF2 to RPI-RP2. which will do a 'deep clean' on your Flash Memory. **You will lose all the files on the board**, but at least you'll be able to revive it! After loading this UF2, follow the steps above to re-install CircuitPython.

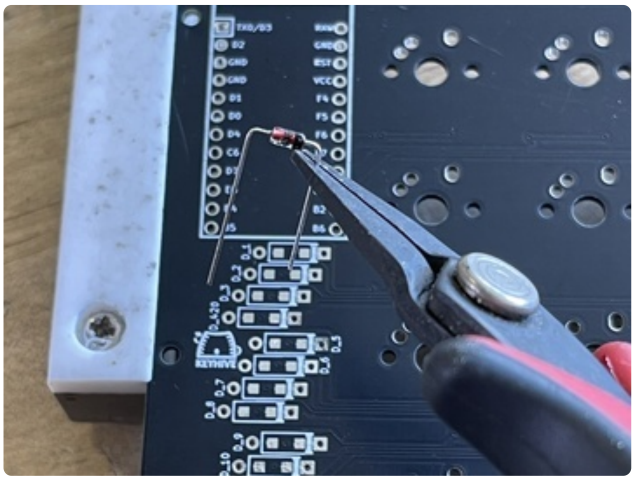
Download flash erasing "nuke" UF2

https://adafru.it/RLE

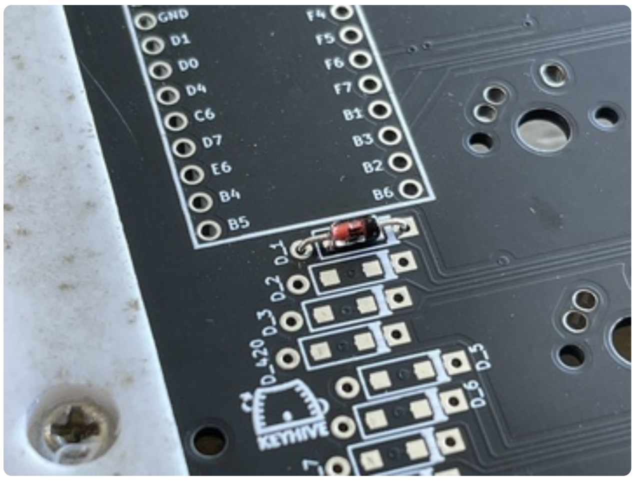
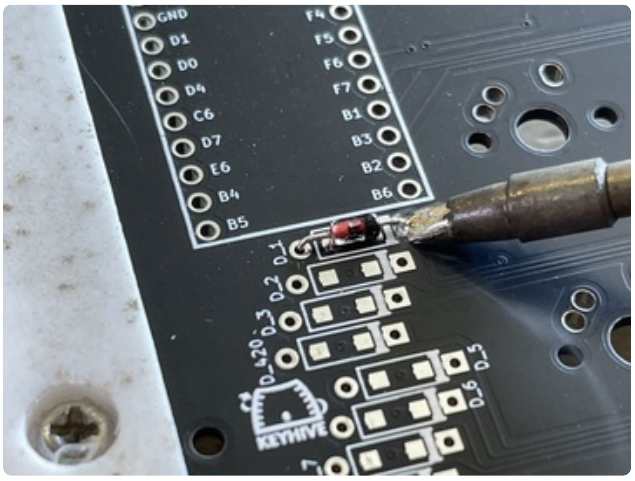
Assembly



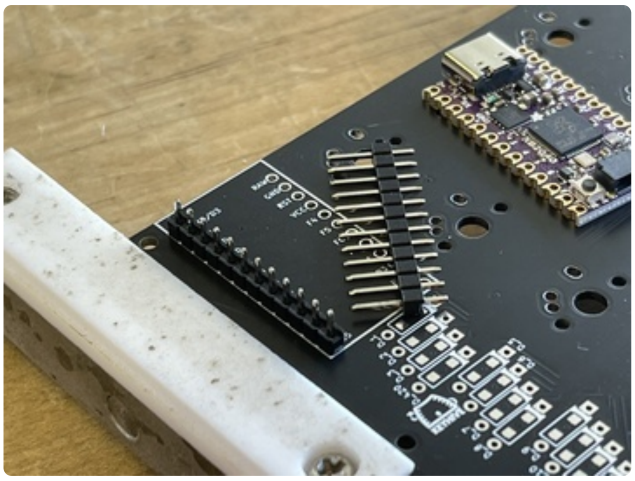
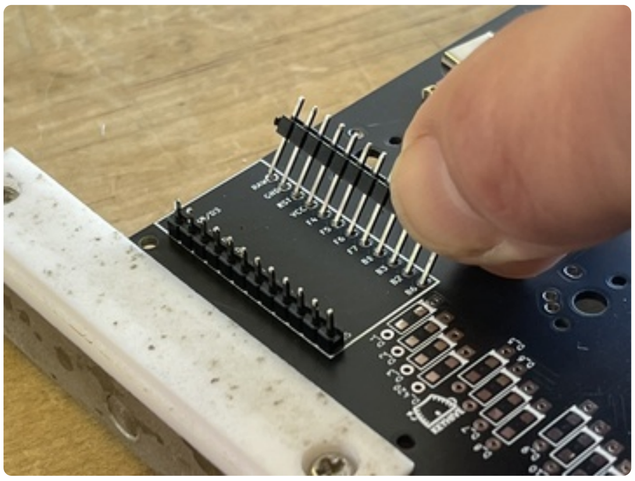
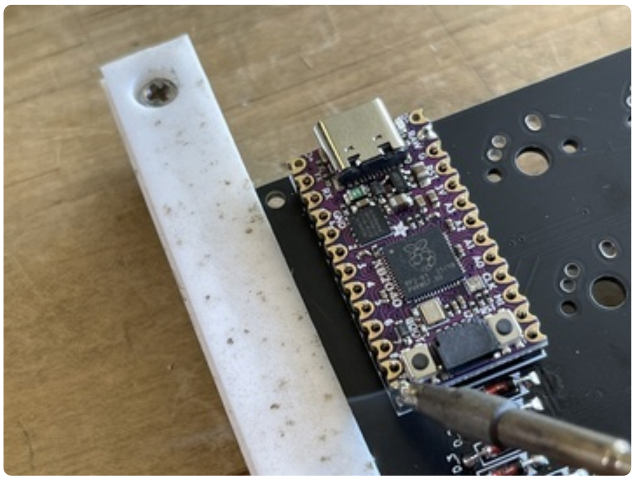




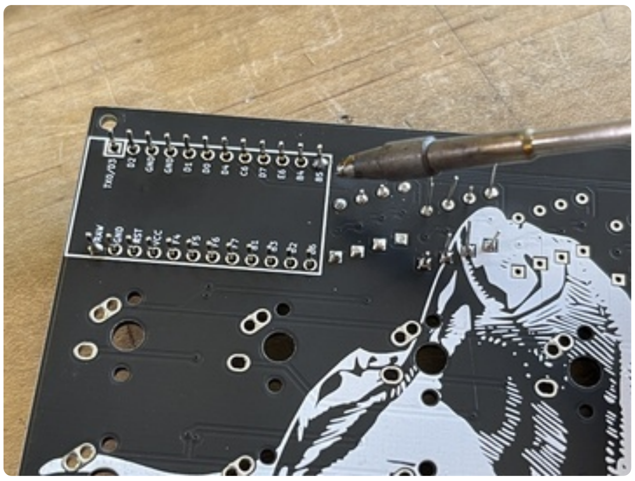
The first thing you'll want to do is bend the diodes. This is very easy to do with a small pair of pliers. The ones used in the picture are these (http://adafru.it/1368).

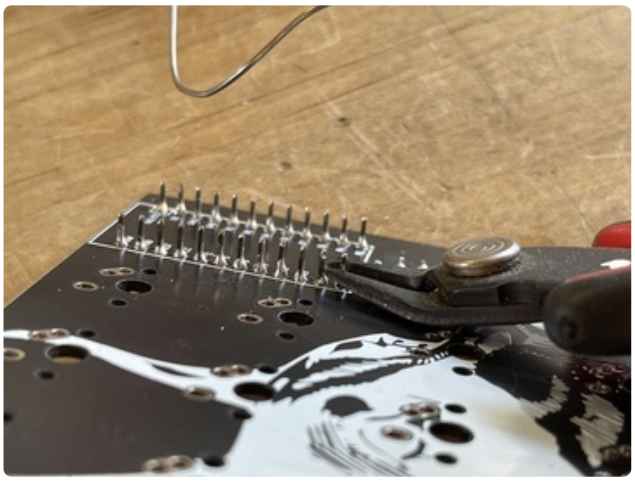
Next, solder in all the diodes. Make sure the black band is on the side of the square soldering pad. After soldering all the diodes, go over to the other side and cut off the excess legs.

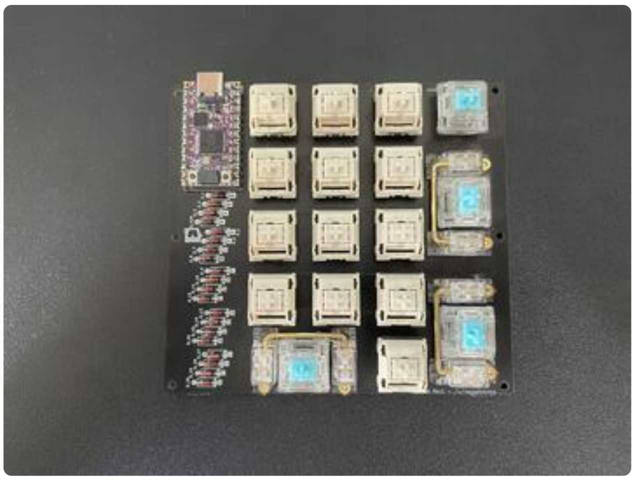




Now, put the headers in long-side facing downwards. Place the KB2040 on top with the microcontroller side facing outwards and solder it to the headers.



After that, flip the board over and solder the headers to the PCB. You can cut off the excess leg length if you'd like to, but doing so isn't necessary.





Before assembling the case, attach the stabilizers and switches.

Finally, attach the two halves of the case with the standoffs and screws included in the kit and put the keycaps on.

KMK Installation Quick Start

KMK Installation Quick Start (https://adafru.it/19Oa)

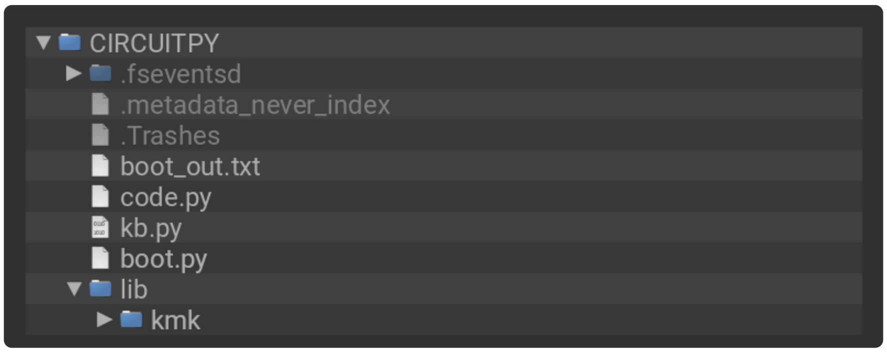
Setting Up KMK

After having installed KMK on the previous page, you're going to want to click 'Download Project Bundle' on the file below. Unzip the file, copy **kb.py** to your **CIRCUITPY** drive and rename **main.py** to **code.py** and copy it to the **CIRCUITPY** drive as well.

```
# SPDX-FileCopyrightText: 2022 Eva Herrada for Adafruit Industries
# SPDX-License-Identifier: MIT

import board

from kmk.kmk_keyboard import KMKKeyboard as _KMKKeyboard
from kmk.matrix import DiodeOrientation


class KMKKeyboard(_KMKKeyboard):
    row_pins = (board.D5, board.D6, board.D7, board.D8, board.D9)
    col_pins = (
        board.A1,
        board.A0,
        board.SCK,
        board.MISO,
    )
    diode_orientation = DiodeOrientation.COLUMNS
    i2c = board.I2C
```

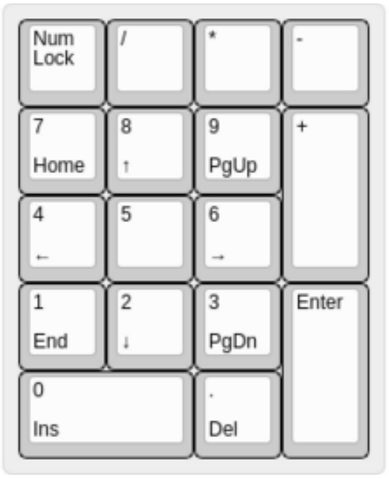
Change line from kmk.matrix import DiodeOrientation to kmk.scanners import DiodeOrientation per latest documentation

After you've copied everything over, your **CIRCUITPY** drive should look something like this.

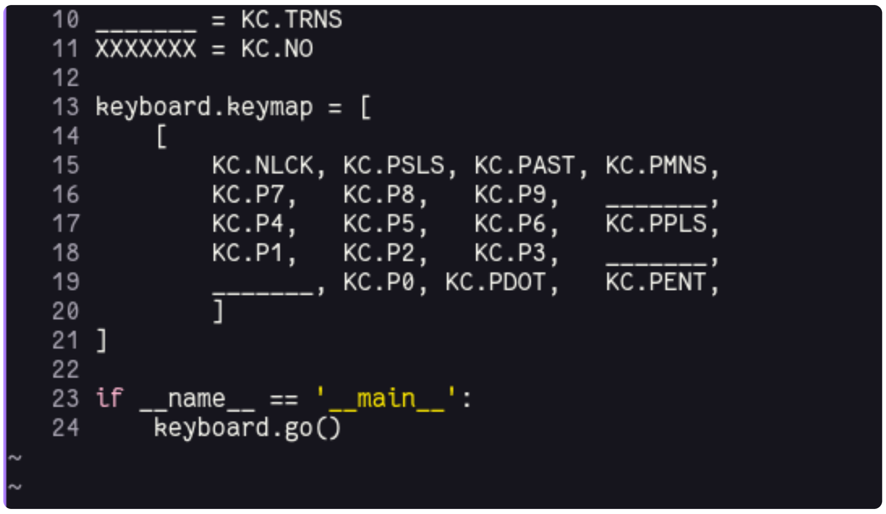


```
▼ CIRCUITPY
  ▶ .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
    kb.py
    boot.py
  ▼ lib
    ▶ kmk
```

Key Mapping



The default keymap for this macropad is the numpad keymap.



```
_______ = KC.TRNS
XXXXXXX = KC.NO

keyboard.keymap = [
    [
        KC.NLCK, KC.PSLS, KC.PAST, KC.PMNS,
        KC.P7,   KC.P8,   KC.P9,   _______,
        KC.P4,   KC.P5,   KC.P6,   KC.PPLS,
        KC.P1,   KC.P2,   KC.P3,   _______,
        _______, KC.P0, KC.PDOT,   KC.PENT,
        ]
]

if __name__ == '__main__':
    keyboard.go()
```

At this point, the keyboard should just work when you hit the keys. Feel free to mess around with the key maps. KMK has some good documentation on this process. (https://adafru.it/1a0W)