Mini-Guide: Getting Started With The WiPy Board

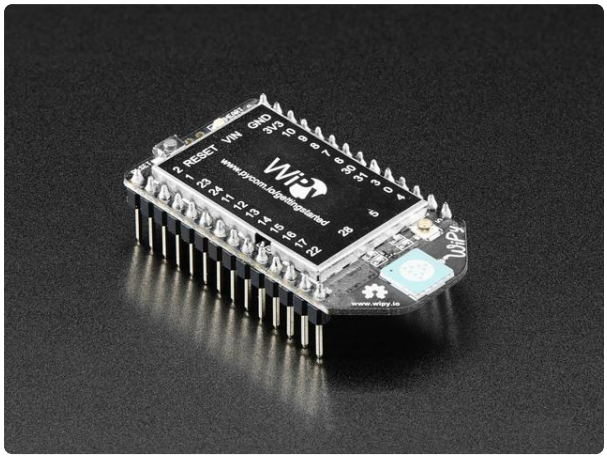
Created by Philip Moyer

https://learn.adafruit.com/micro-guide-getting-started-with-wipy-board

Last updated on 2022-12-01 02:47:40 PM EST

Table of Contents

Overview



WiPy 1.0 - IoT Development Platform
The WiPy is an enterprise grade IOT development platform
that runs Python in real time with the perfect blend of power, speed, friendliness, and flexibility....
https://www.adafruit.com/product/3184

The WiPy is a microcontroller that runs MicroPython on a light-weight processor along with a WiFi chip so the device can communicate with wireless networks. As a matter of fact, the device pretty much must communicate with wireless networks since that's how you interact with them.

The WiPy is one of a growing selection of possible MicroPython boards that you can use for experimenting, programming, and/or Internet of Things development (but more on security later). The big ones seem to be the WiPy (http://adafru.it/3184), the PyBoard (http://adafru.it/2390), and the ESP8266-based boards (such as the Adafruit Feather Huzzah! ESP8266 (http://adafru.it/2821) or the Huzzah! breakout (http://adafru.it/2471)), though the microBit is likely going to be a player, particularly in the UK, because of its deployment footprint.

This guide is about the WiPy, but for information about the overall MicroPython ecosystem check out Tony DiCola's excellent series of guides - there's a list on the "Additional Resources" page (). Also, a big shout-out to Tony for the fantastic work on MicroPython, including his guides and streaming videos.

The WiPy can be purchased and used on its own without anything but a USB cable for power. There is also a no-soldering expansion board that includes a micro-USB connector, a 2-pin JST for a LiPo battery, the necessary LiPo charging circuitry, and a micro-SD card slot for storage expansion, on top of the GPIO pin breakouts.

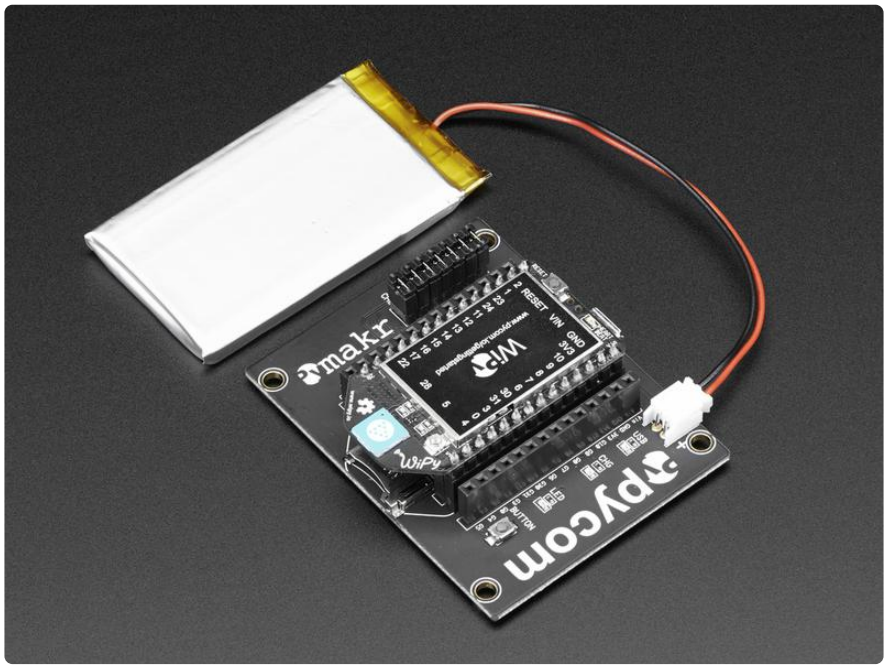
While you're ordering the WiPy and the expansion board you can pick up a LiPo battery of an appropriate size for your use (). I use a 6600 mAh three-cell pack from Adafruit (http://adafru.it/353) and I honestly haven't run it out of power yet. I'll test it to see how long it will run with that battery, but it's going to be a good long time.

When the LiPo is plugged in and the USB is connected to a power source, the onboard charging circuit will keep the LiPo topped off. The LiPo battery support is handy for when you want to take your project mobile!

Default Configuration

Construction of the WiPy board is trivial.

1. Insert the WiPy board into the expansion board. The pins will go in the center row of female headers! Follow the silkscreen outline to get the board inserted properly; the reset button on the WiPy should be over the USB connector on the expansion board.
2. Plug the LiPo battery into the JST connector.
3. Plug a micro-B USB cable into the expansion board and then plug it into a power source. I have mine connected to a powered USB hub, but your laptop should provide enough power, as will a wall adapter or multi-port USB charger like the Anker.

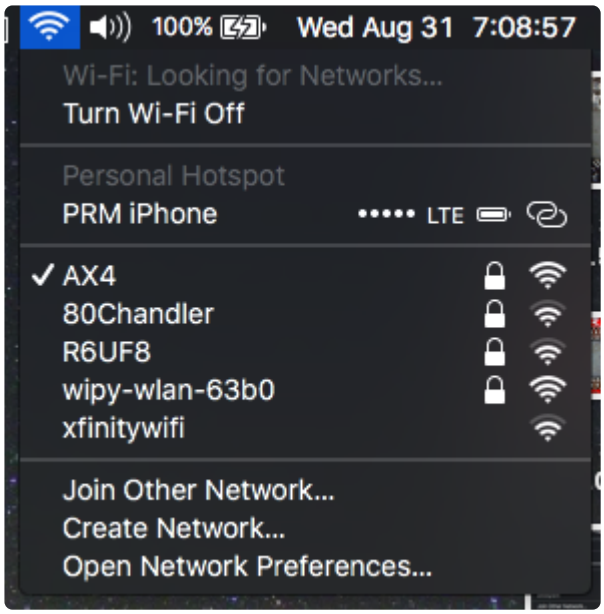


That's it. Your WiPy will be up and running, ready for programming! Notice that MicroPython comes pre-flashed on the WiPy board, unlike the ESP8266, so you don't have to do anything to install it.

The WiPy board boots into WiFi AP mode, so it creates its own WiFi network when it starts up. This is part of the factory firmware. The WiPi will be at the IP address 192.16 8.1.1 The WiFi network will be named wipy-wlan-something where "something" is a

small set of hex numbers. Connect your laptop's WiFi to that network using the password www.wipy.io (I have not found a way to change this default)

On Mac OS X, click on the WiFi icon in the toolbar. The drop-down will give you a list of available WiFi networks. Click on the one that starts with wipy-wlan-...



The system will then bring up a dialog box for you to enter the WiFi password. This will be "www.wipy.io".

Then just click "Join". Windows and Linux have similar methods for attaching to new WiFi networks. Remember, you need to do this on your laptop, which is where you will be editing the configuration files!

Once you have connected to the WiPy's network, you can access the device. To get to the Read-Evaluate-Print Loop (REPL) prompt on the board, telnet to 192.168.1.1 You will be prompted to log in. The login is micro and the password is python

While this is the default, it can be changed in the boot sequence. See the image below for an example of a successful telnet connection to the WiPy REPL.

Telnet is a program that connects to a remote system and emulates a terminal so you can log in on and interact with the system remotely. It was the predecessor to SSH. Telnet has fallen into disuse because it has security exposures that cannot be fixed. It originated with UNIX systems in the late 1970's and early 1980's. It was how we used to connect to remote systems.

Mac OS X comes with a telnet client (and an ftp client), so you don't have to do anything special to install one on those systems.

On Linux systems, you use your local package manager (which is distro-dependent) to install the ftp and telnet packages.

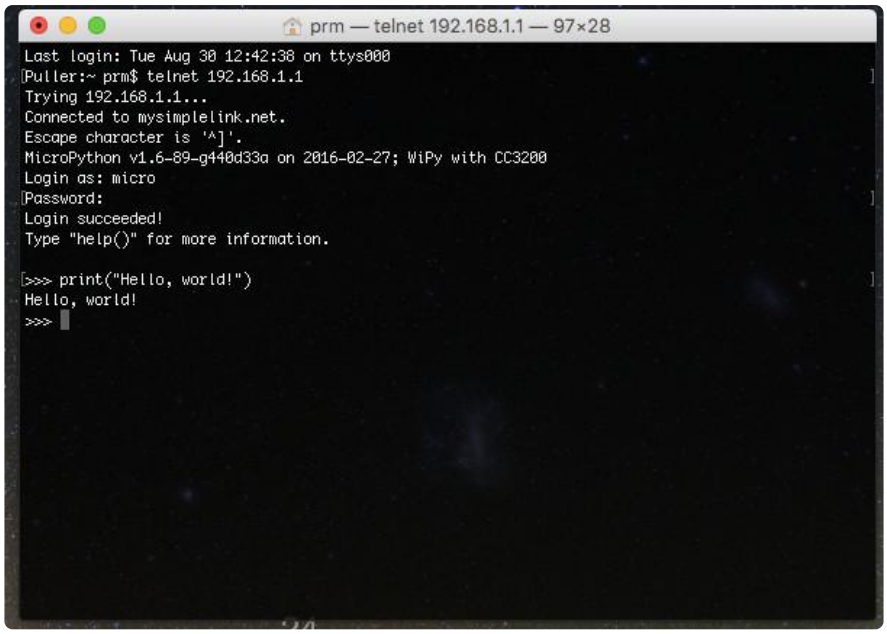
On Windows, things are a bit more complicated. There is a Microsoft TechNet article on installing telnet (). For ftp, the most popular solution is the free FileZilla program (). Download and install the client application.

MicroPython on the WiPy does not support floating point processing! Code that generates a float will throw an error.

After you log in you will see the Python `>>>` prompt. The system is now ready for you to type Python code into it. As of this writing, the board ships with MicroPython v1.6-89-g440d33a.

MicroPython is Python3, so Python2 code will need to be converted to Python3 in order to run. This isn't a big deal if you are writing code from scratch but some older code might need slight adjustments

Here is a screen shot of the `telnet` session showing the connection process and successful REPL execution:



Modifying boot.py

There is a small file system in the flash memory on the WiPy. This contains all the support code, the files run at boot time, and any code the user uploads to the board (unless you use an SD card, which I highly recommend).

The boot sequence when you power on or reset the board is

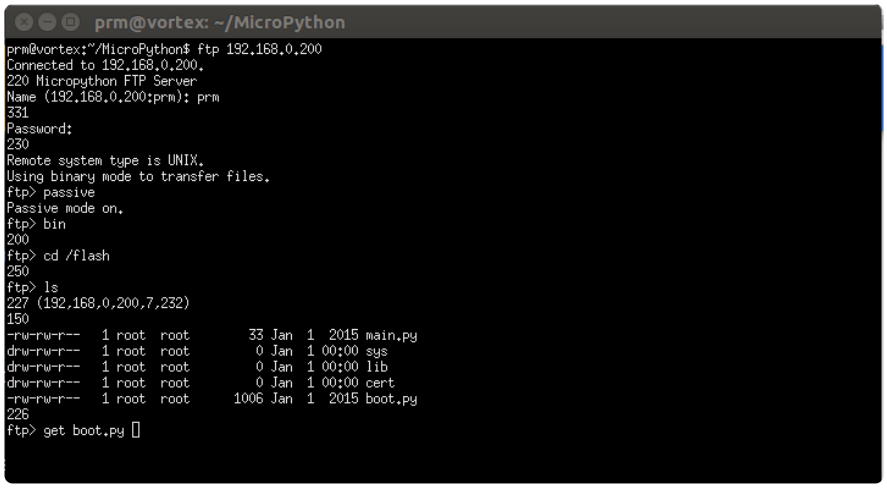
1. Power on
2. Bring up the Python interpreter (REPL)
3. Execute the boot.py program
4. Execute the main.py program
5. Start the telnet and ftp servers

If you want to change the network or server configurations (like the username and password), you edit the boot.py program.

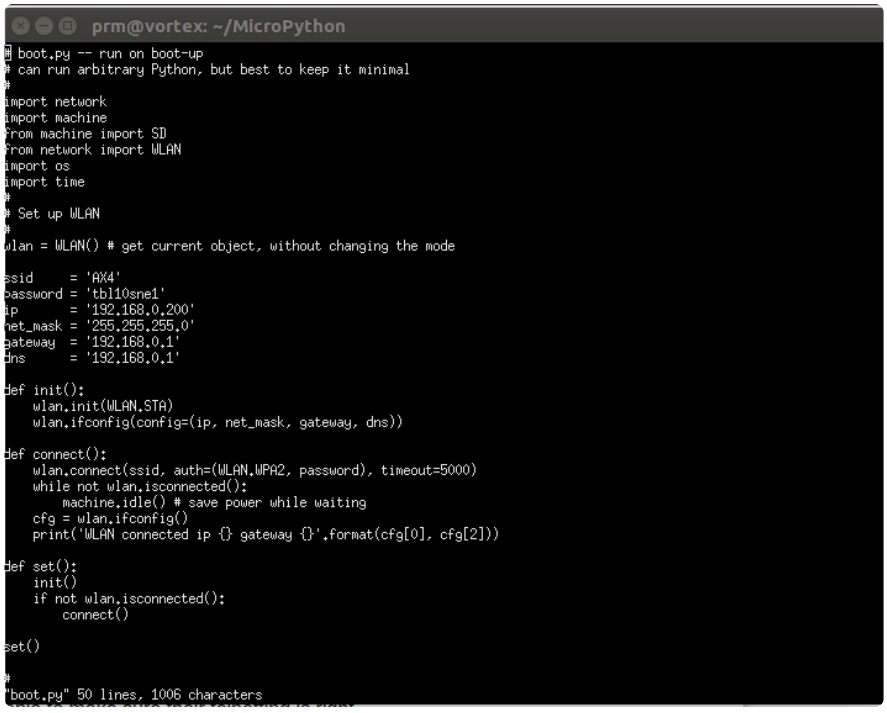
To edit the boot.py program, you first need to get it from the WiPy board. You do this using the ftp program (ftp stands for File Transfer Protocol).

- When you connect using ftp, you will be prompted for a login and password.
- The default login is micro and the default password is python
- After you have connected, you need to enter the passive mode and the binary mode. Type `passive` and press Enter. Then type `bin` and press Enter.
- Then change to the /flash subdirectory by typing `cd /flash` and pressing Enter.

- Then just type `get boot.py` to copy the boot.py template to your local computer.



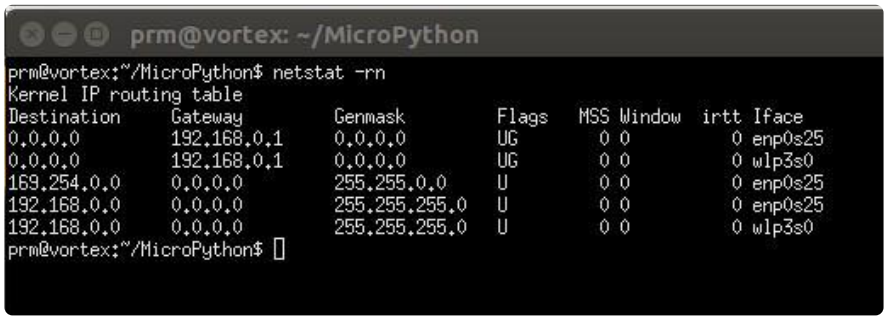
Then edit your local copy of boot.py using your favorite editor.



If you make a mistake in boot.py or main.py you will likely brick the device. This guide has instructions for fixing it.

Here is my boot.py program. You will need to change the configuration to match your local network. Note that this assumes yu are assigning a static IP address to the device! I also assume you have a micro-SD card (FAT formatted) in the SD card slot. You really want an SD card for extra storage!

On UNIX systems (Linux and Mac OS X) and Windows you can find appropriate network configurations by using the netstat command. Open a terminal or the command prompt and enter `netstat -rn`



You want to take the gateway address from the 0.0.0.0 destination, so in this case, the gateway would be 192.168.0.1. Take the netmask value from your local wireless network; in this case, that is 192.168.0.0, so the appropriate netmask is 255.255.255.0. The DNS server is usually the IP address of your wireless access point (in this case, 192.168.0.1) but if you don't know it or aren't sure, 8.8.8.8 is always a safe bet - that's Google's open DNS server.

The only other thing yo need to do is find an open IP address to use on your local network. I keep a list of all used IP addresses in my network's /etc/hosts file. Another way to find an open IP address is to log into your wireless access point and look for all the assigned DHCP addresses and pick one that's "far" away from any assigned addresses. On my network, I have everything between .2 and .100 assigned to various devices, like my huge 158-core cluster. DHCP assigns dynamic addresses between .100 and .150, so I picked .200 for my WiPy. Remember that .254 is the maximum value for any address component (called an "octet").

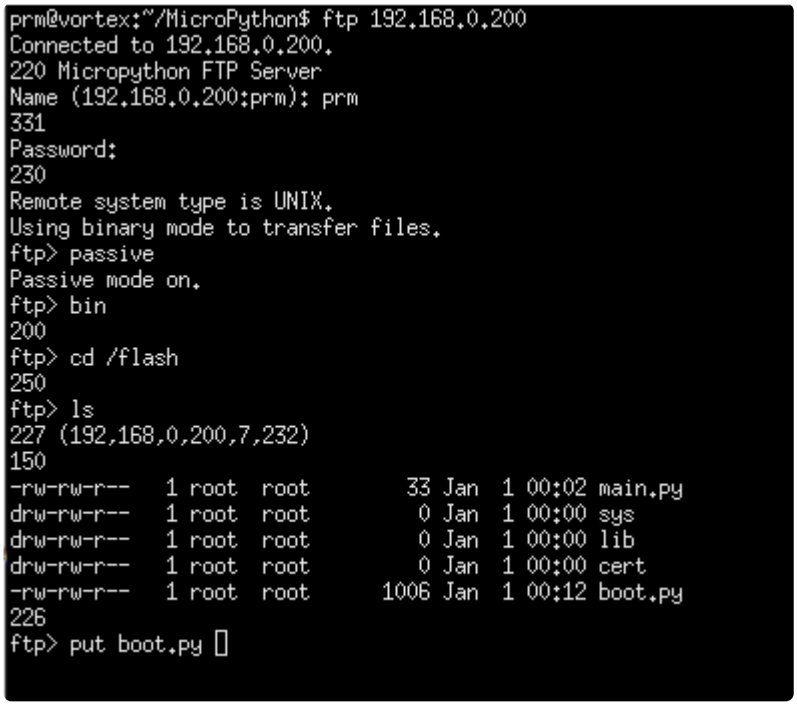
Note: it is possible to use DHCP to get an address, but the WiPy board will not register with DNS for a .local address. To get to the board you have to use some method to find out the IP address or use a fairly complex method to enable the REPL on the UART interface and connect to the board over USB (which is beyond the scope of this intro guide).

```
# boot.py -- run on boot-up
# can run arbitrary Python, but best to keep it minimal
#
import network
import machine
from machine import SD
from network import WLAN
import os
import time
#
# Set up WLAN
#
wlan = WLAN() # get current object, without changing the mode
```

```
ssid     = 'YOUR_SSID'
password = 'YOUR_WIFI_PASSWORD'
ip       = 'YOUR_STATIC_IP_ADDRESS'
net_mask = 'YOUR_NETMASK'
gateway  = 'YOUR_GATEWAY_IP_ADDRESS'
dns      = 'YOUR_DNS_SERVER'

def init():
    wlan.init(WLAN.STA)
    wlan.ifconfig(config=(ip, net_mask, gateway, dns))

def connect():
    wlan.connect(ssid, auth=(WLAN.WPA2, password), timeout=5000)
    while not wlan.isconnected():
        machine.idle() # save power while waiting
    cfg = wlan.ifconfig()
    print('WLAN connected ip {} gateway {}'.format(cfg[0], cfg[2]))

def set():
    init()
    if not wlan.isconnected():
        connect()

set()

#
# Set up server
#
server = network.Server()
server.deinit()
server.init(login=('YOUR_USERNAME','YOUR_PASSWORD'),timeout=600)
#
# SD card support
#
sd = SD()
os.mount(sd, '/sd')
```

The WiPy does not support SSH, SFTP, or HTTPS! This means that any connections to the device will send your login and password in cleartext. Use a disposable password!

Once you have modified boot.py to match your local network configuration, use ftp to put the program in the /flash directory on the WiPy.. The server only supports passive mode connections. Also, the binary mode file transfer should be used when transferring files. Here is a screenshot that illustrates the connection:



The commands you need to enter after you've made an ftp connection are:

- login
- password
- `passive`
- `bin`
- `cd /flash`
- `put boot.py`
- `quit`

Once you have uploaded the new boot.py program, press the reset button on the WiPy to execute the new configuration program. Your WiPi should now be on your local network and accessible with the login and password you set. Try `telnet` to the IP address you assigned and log in with the credentials you specified. You should then be back at the REPL prompt.

If you can't connect and (likely) don't see the flashing heartbeat LED on the WiPy, you may have made a mistake editing the boot.py program. See the page titled How To Recover Your Bricked WiPy for instructions on how to get back into the device.

Security

There are some serious security issues with the WiPy. In summary, they are:

1. No support for encrypted communications like SCP, SSH, SFTP, and HTTPS. As a result, you should not expose this device to the Internet and log into it remotely. The telnet and ftp protocols are unencrypted, so your login and password are sent in clear text. Anyone sniffing the network between you and the WiPy will be able to log in on your WiPy and take complete control of it.
2. If the attacker has physical access to the board, they can do either of two things. First, they can reset the board to factory settings and log in with the default credentials. Second, they can just steal the SD card to get access to programs and data stored there.

These two problems are serious enough that you can use the WiPy as a development environment on a trusted network, but you should not use the board as an IoT platform where it is in an untrusted/uncontrolled environment where you might need to access the device remotely. This might be fixed in a future release of the firmware, but I haven't found a way around them yet. Use care and carefully consider risk when you use these devices.

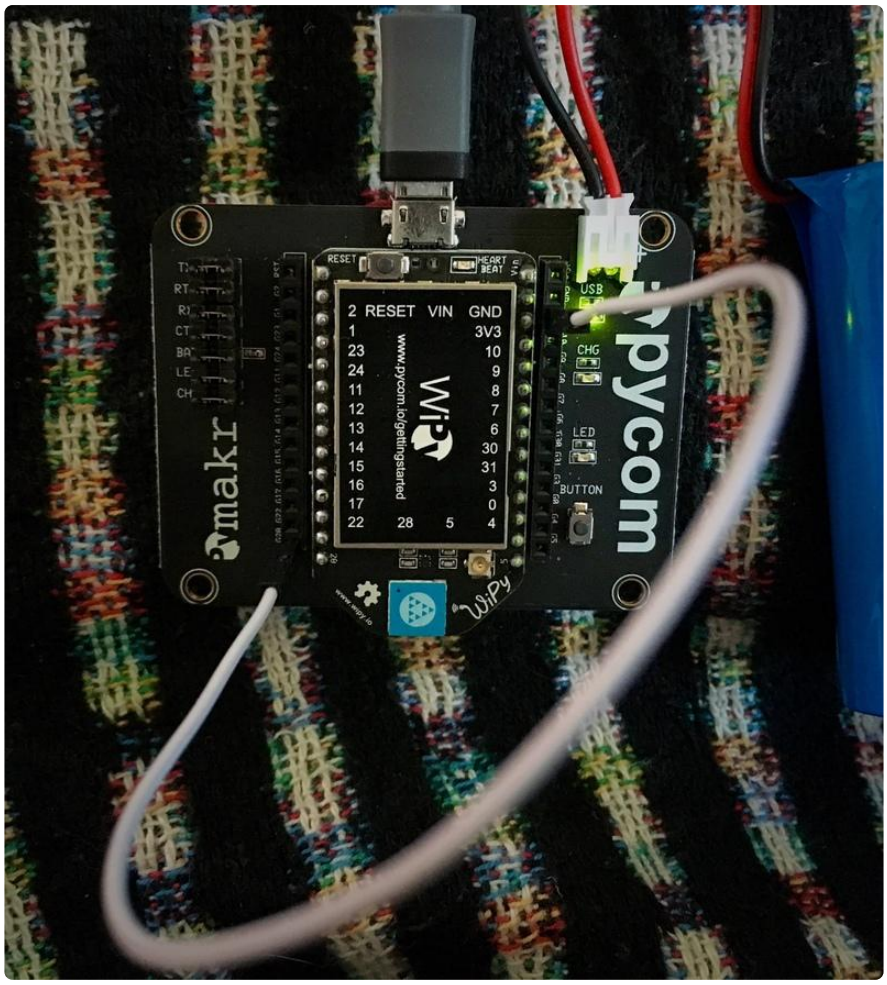
How To Recover Your Bricked WiPy

As noted elsewhere, if you make a mistake in boot.py you can render the board useless (also called "bricking" the device.) Fortunately, there's an easy way to fix it! All it takes is a single jumper wire.

The WiPy stores three firmware configurations: the current firmware config, the previous firmware configuration, and the factory configuration.

Power off the device. Bridge pin G28 and 3V3 with the jumper wire to pull G28 high. Power on the board. The heartbeat LED will start flashing. It will flash for three seconds. For the next three seconds, it will flash faster. For the next three seconds, it will flash even faster. After that, disconnect the jumper and the board will be running the factory firmware without running either boot.p or main.py!

At this point you can connect your laptop to the default WiFi network that the WiPy generates, `ftp` to 192.168.1.1, and put a fresh, unbroken version of boot.py in flash. Push reset and the WiPy will boot with the new configuration. It's that simple!



Additional Resources

Here are some additional resources you can use to learn more about MicroPython:

- Tony DiCola's excellent series on MicroPython and the ESP8266:
- MicroPython Basics: What is MicroPython? ()
- MicroPython Basics: ESP8266 WebREPL ()
- MicroPython Basics: Load Files & Run Code ()
- Building and Running MicroPython on the ESP8266 ()
- MicroPython Basics: Blink a LED ()
- MicroPython Basics: How to Load MicroPython on a Board ()
- The official MicroPython website ()
- MicroPython documentation ()