IoT-ALE:
Discovering Tiny Snakes

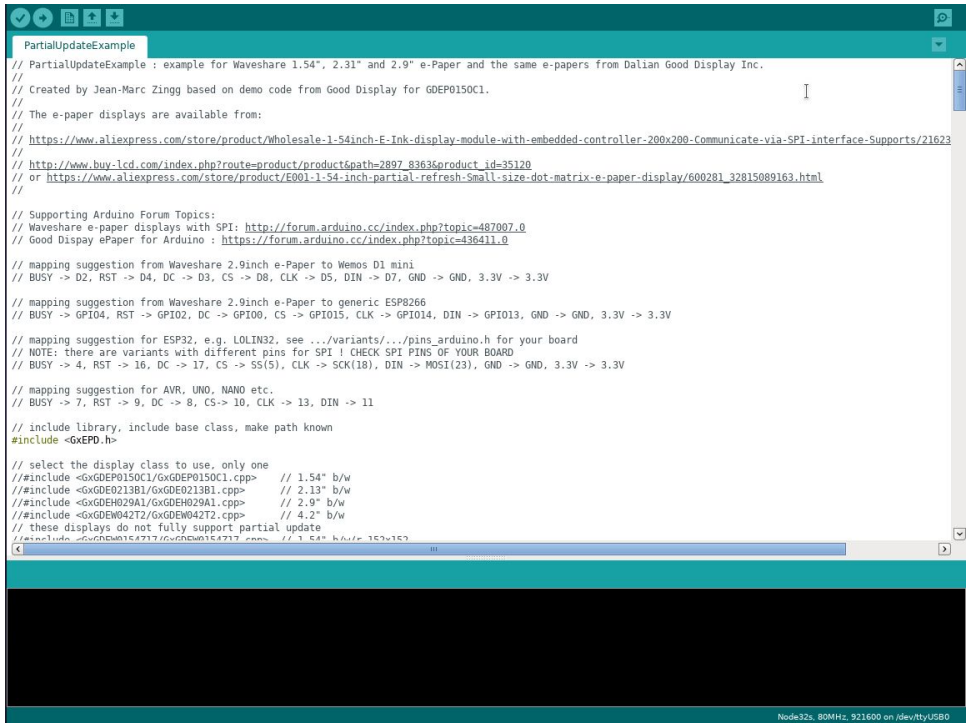
IoT development without the need to compile
(mostly)

John ‘Warthog9’ Hawley

SCaLE 17x - March 2019

Quick: MicroPython vs. CircuitPython?

Why is this different?



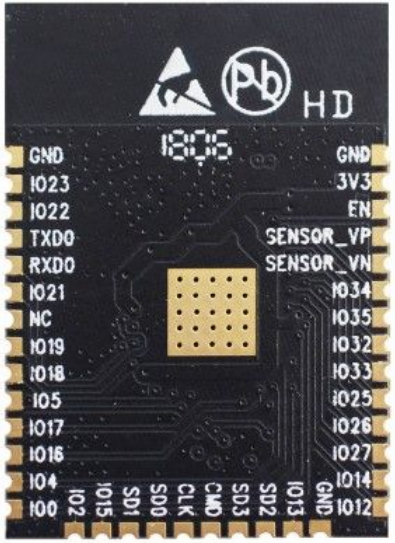

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:4732
load:0x40078000,len:7496
load:0x40080400,len:5512
entry 0x4008114c
I (388) cpu_start: Pro cpu up.
I (389) cpu_start: Single core mode
I (389) heap_init: Initializing. RAM available for dynamic allocation:
I (392) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (398) heap_init: At 3FFC4F48 len 0001B0B8 (108 KiB): DRAM
I (405) heap_init: At 3FFE0440 len 00003BC0 (14 KiB): D/IRAM
I (411) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (417) heap_init: At 40091448 len 0000EBB8 (58 KiB): IRAM
I (424) cpu_start: Pro cpu start user code
I (218) cpu_start: Starting scheduler on PRO CPU.
Setting up LEDs
Setting up Buttons
Setting up Sensor I2C
Setting up BME280
Setting up TSL2591
bme_values[0]: 2172 - 21.72C
bme_values[1]: 25929420 - 1012.86hPa
bme_values[2]: 44558 - 43.51%
tsl_values[0]: 48
tsl_values[1]: 21
All Good
Initialize the Board LED as a PWM... Success
To break hit <ctrl>+c then enter: breathTimer.deinit()
OSError: [Errno 2] ENOENT
MicroPython v1.9.4-560-g185716514 on 2018-09-20; ESP32 module with ESP32
Type "help()" for more information.
>>>
```

Why is this different?

- Quick, iterative, development
- Most of the advantages of Python
- 0 to blinking LED very quick
- Mostly no need to compile anything
- Lots of default functionality, and upip (library / package management!)

Why is this possible?

- Same reason IoT is becoming ubiquitous
  - MCUs & CPUs are getting more powerful, and cheaper
- ESP32 on the SensorNode cost $5.10 to place on the board.
  - Dual Core
  - Wifi (802.11b/g/n up to 150Mbps 2.4GHz)
  - Bluetooth (v4.2 BR/EDR & BLE)
  - 4MB of flash
  - 520KB RAM
- There’s lots of competition in this space



Flashing MicroPython:

With the VM:

- Select the VM, plug in SensorNode
  - Should cause it to attach to the VM, if it's not *VM -> Removable Devices* and attach it
- Helper script (specific to this tutorial)

  flash_sensornode.sh
  - Sets Serial port (usually /dev/ttyUSB0)
  - Fully erases the flash on the ESP32
    - *esptool.py --chip esp32 --port "${USBPORT}" erase_flash*
  - Flashes MicroPython
    - *esptool.py --chip esp32 \ --port "${USBPORT}" --baud 460800 \ write_flash -z 0x1000 "${flash_file}"*

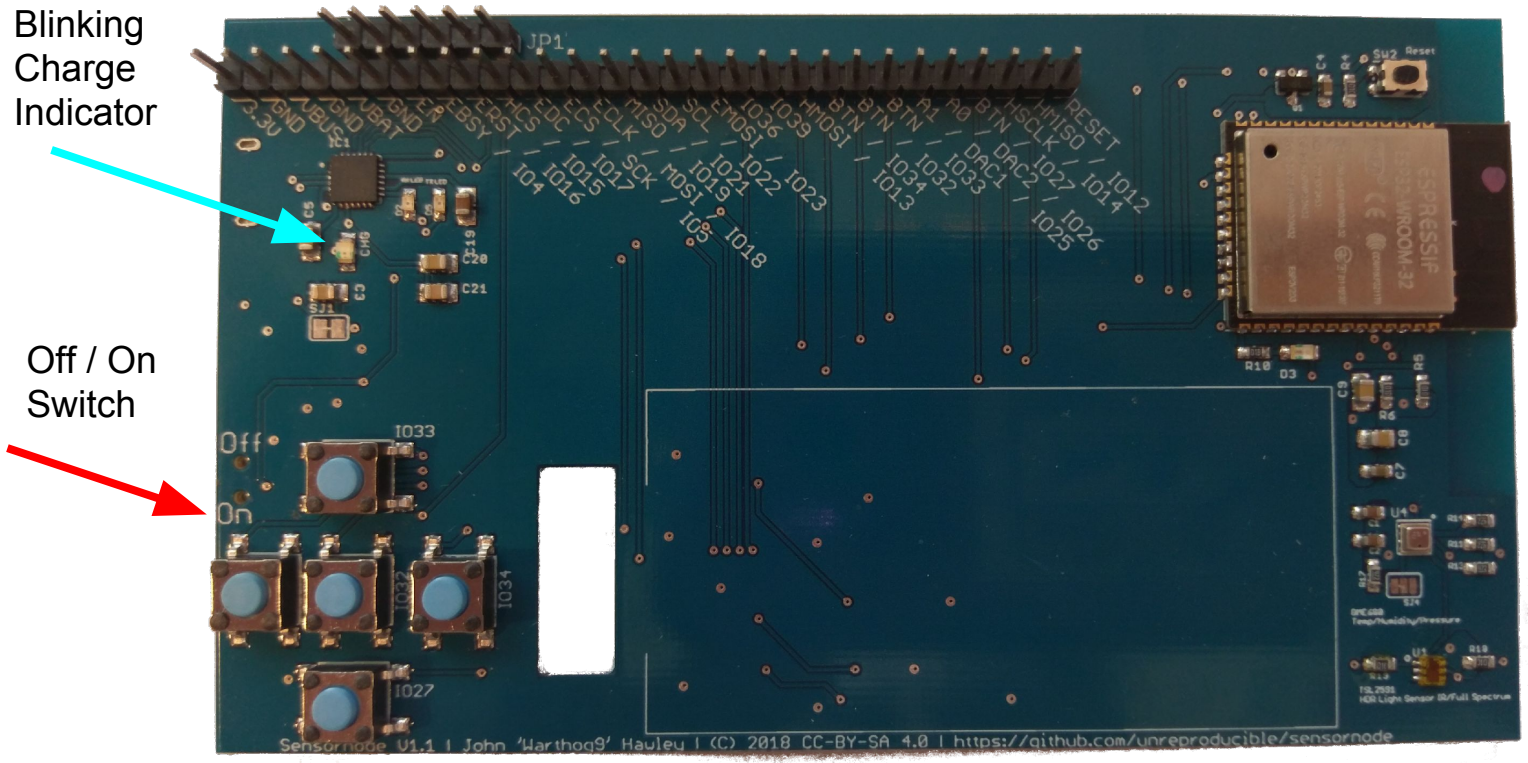
Without the VM:

- Serial Drivers
  - Linux: Driver in Most Distros
  - Windows / Mac: Install Silicon Mechanics CP2104 https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers
- Download / Install esptool
  - This requires Python
  - Linux: distro packages are available
  - Windows / Mac: use pypi to install
- Download MicroPython & Upload it to the board
  - http://micropython.org/download#esp32
  - esptool.py --chip esp32 \ --port /dev/ttyUSB0 erase_flash && \ esptool.py --chip esp32 --port \ /dev/ttyUSB0 write_flash -z 0x1000 \ <path to micropython .bin>

Make Sure the SensorNode is ‘on’

Blinking Charge Indicator

Off / On Switch



Helpful tip:

If there’s a flashing light on the board it’s on (it’s the charging indicator light).
If it’s solid, it’s off.

The switch is on the side with the USB port:

- Down = On
- Up = Off

Breaking down the flash commands

```
esptool.py \
        --chip esp32 \                  # Identifies which chip variant we are dealing with
        --port /dev/ttyUSB0 \           # Identifies which port the serial device is on
        erase_flash \                   # Erases the flash area of the chip
&& \                                    #     (not including the boot loader area)
esptool.py \
        --chip esp32 \                  # Identifies which chip variant we are dealing with
        --port /dev/ttyUSB0 \           # Identifies which port the serial device is on
        write_flash \                   # Indicates to write to the flash chip
        -z 0x1000 \                     # Indicates WHERE on the flash chip to write to
        <path to micropython .bin>      # What to flash to the chip
```

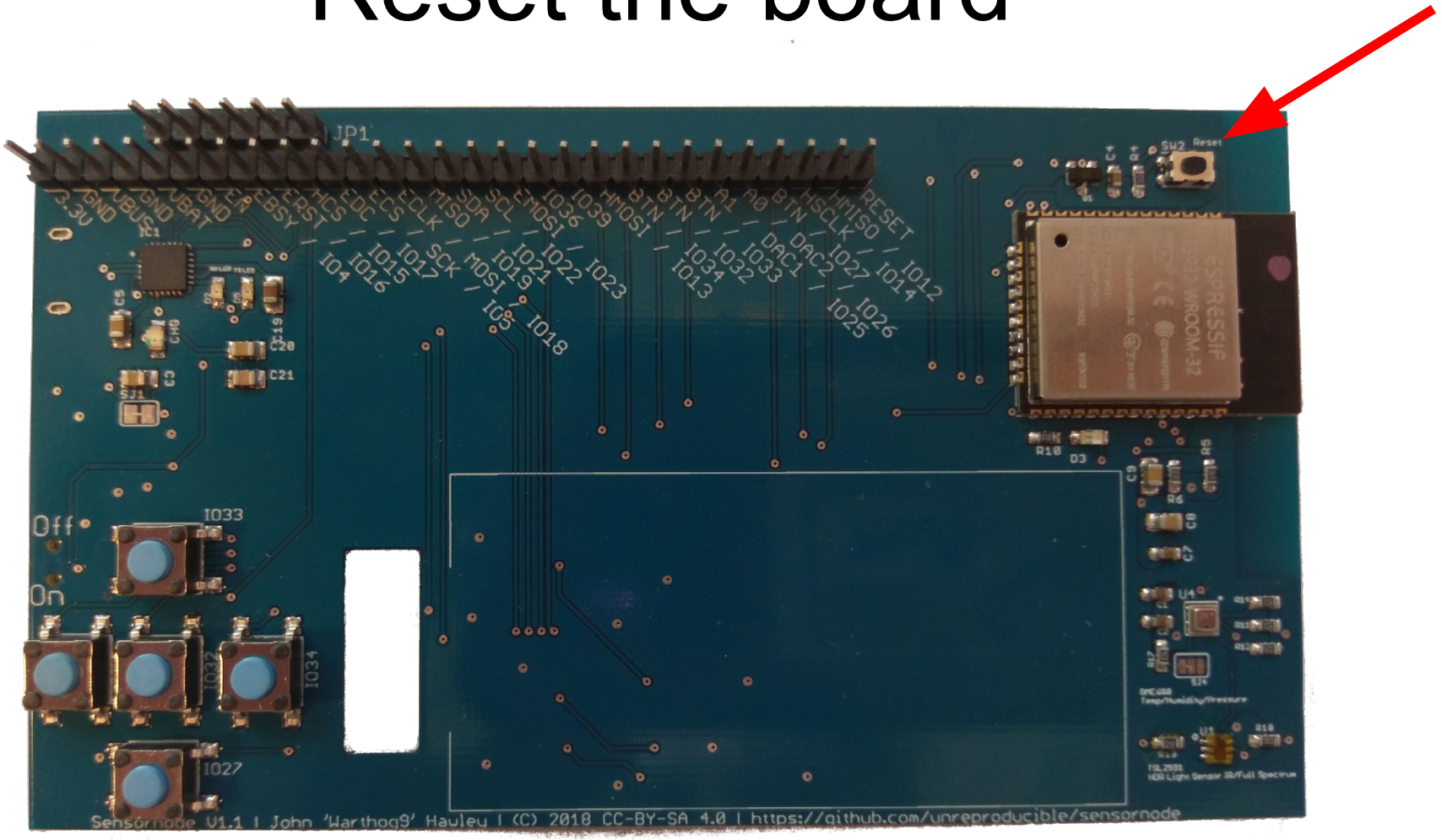
What this should look like:

```
[root@tutorial-base ~]# dmesg | tail -n 8
[...]
[86344.904683] cp210x 2-2.1:1.0: cp210x converter detected
[86344.915286] usb 2-2.1: cp210x converter now attached to
ttyUSB0
[root@tutorial-base ~]# ./flash_sensornode.sh
Flash File: esp32-20190214-v1.10-98-g4daee3170.bin
esptool.py v2.7-dev
Serial port /dev/ttyUSB0
Connecting.....
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, Coding Scheme None
MAC: 30:ae:a4:86:c7:64
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 4.4s
Hard resetting via RTS pin...
esptool.py v2.7-dev
Serial port /dev/ttyUSB0
Connecting......
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, Coding Scheme None
MAC: 30:ae:a4:86:c7:64
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 1133232 bytes to 714809...
Wrote 1133232 bytes (714809 compressed) at 0x00001000 in
18.6 seconds (effective 488.0 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
[root@tutorial-base ~]#
```

Open up the serial console

- Minicom:
  - minicom -D /dev/ttyUSB0 --baudrate 115200
    (to exit <ctrl>c-q)
- Screen:
  - screen /dev/ttyUSB0 115200n8
    (to exit <ctrl>c-A \)
- Windows: use PuTTY

Reset the board



On the serial console...

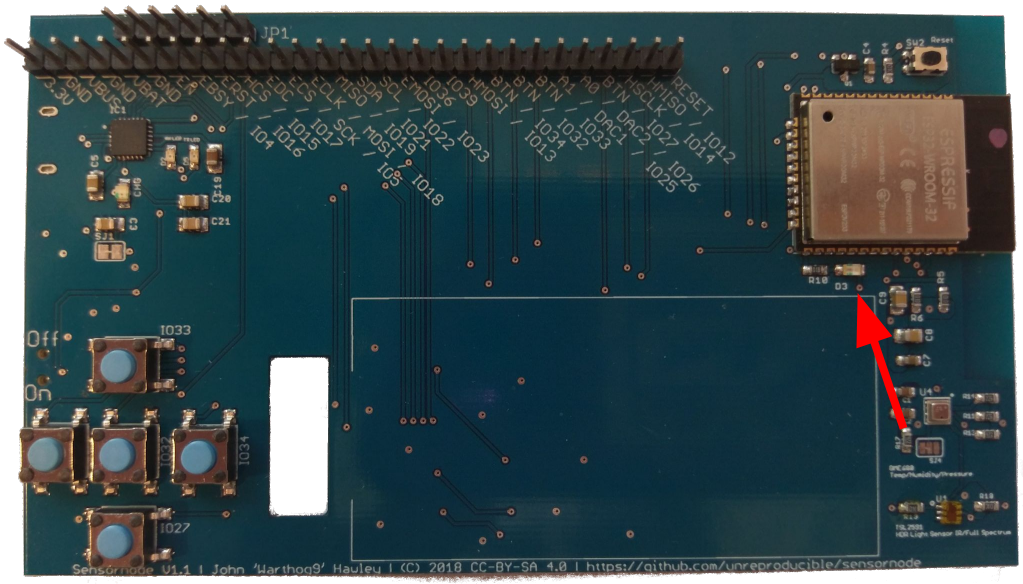
```
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:5060
load:0x40078000,len:8788
ho 0 tail 12 room 4
load:0x40080400,len:6772
entry 0x40081610
I (428) cpu_start: Pro cpu up.
I (428) cpu_start: Application information:
I (428) cpu_start: Compile time:     12:32:34
I (430) cpu_start: Compile date:     Feb 14 2019
I (436) cpu_start: ESP-IDF:          v3.3-beta1-268-g5c88c5996
I (442) cpu_start: Single core mode
I (447) heap_init: Initializing. RAM available for dynamic allocation:
I (454) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (460) heap_init: At 3FFB92B0 len 00026D50 (155 KiB): DRAM
I (466) heap_init: At 3FFE0440 len 0001FBC0 (126 KiB): D/IRAM
I (472) heap_init: At 40078000 len 00008000 (32 KiB): IRAM
I (479) heap_init: At 40092834 len 0000D7CC (53 KiB): IRAM
I (485) cpu_start: Pro cpu start user code
I (55) cpu_start: Starting scheduler on PRO CPU.
OSError: [Errno 2] ENOENT
MicroPython v1.10-98-g4daee3170 on 2019-02-14; ESP32 module with ESP32
Type "help()" for more information.
>>>
```

Quick *Hello World!*

```
>>> print("Hello World!")
Hello World!
>>>
```

Now to Blink an LED!

```
>>> import machine
>>> led_pin = machine.Pin(0, machine.Pin.OUT)
>>> led_pin.on()
>>> led_pin.off()
```



Note: You'll quickly find the on() turns the LED off, and off() turns the LED on. To "Fix"

```
>>> led = machine.Signal( led_pin, invert=True)
>>> led.off()
>>> led.on()
```

Some interesting things to note

- boot.py
  - executed on every start, good for setting up the board (good place for wifi settings for example)
- main.py
  - Run after boot.py, think of it like the autoexec.bat
- It’s possible to upload more files to the board
  - Ampy - https://github.com/adafruit/ampy
- Tab completion works in the repl prompt
- <ctrl>+e at the repl prompt puts you into “paste” mode

Disconnect From Serial before trying file transfers!

- Minicom:
  - to exit: <*ctrl*>*c-q*
- Screen:
  - to exit: <*ctrl*>*c-A \ y*
- Putty:
  - Hit the X and close the application

Where to go from here

Setup Wifi in client mode

- ampy --port /dev/ttyUSB0 get boot.py | tee boot.py

```
# This file is executed on every boot (including wake-boot from deepsleep)
#import esp
#esp.osdebug(None)
#import webrepl
#webrepl.start()
```

- Add to boot.py:

```
# This file is executed on every boot (including wake-boot from deepsleep)
#import esp
#esp.osdebug(None)
#import webrepl
#webrepl.start()
import network
sta = network.WLAN(network.STA_IF)
sta.active(True)
sta.connect("ALE", "Penguins")
```

- ampy --port /dev/ttyUSB0 put boot.py

Re-connect to Serial and check:

- >>> sta.ifconfig()
  ('192.168.123.456', '255.255.255.0', '192.168.123.1', '192.168.123.1')
  >>> sta.status()
  1010
  >>> sta.isconnected()
  True
  >>>

- >>> import socket
  >>> addr_info = socket.getaddrinfo("towel.blinkenlights.nl", 23)
  >>> addr = addr_info[0][-1]
  >>> s = socket.socket()
  >>> s.connect(addr)
  >>> while True:
  … data = s.recv(500)
  … print(str(data, 'utf8'), end=’’)
  …
  …
  …
  <ctrl>+c will stop the while loop

One more thing to note, but not try here...

- Access Point Mode (can be used with client mode at the same time, albeit slowly)
  - >>> ap = network.WLAN(network.AP_IF)
    >>> ap.active(True)
    >>> #ap.config(essid="network-name", authmode=network.AUTH_WPA_WPA2_PSK, password="abcdabcdabcd")
  - Can be added to boot.py, same as the client information

Links to more resources

- https://github.com/unreproducible/tinysnakes
- https://docs.micropython.org/en/latest/esp8266/tutorial/intro.html
  (note: most of the ideas are the same, the boards ARE different)
- https://boneskull.com/micropython-on-esp32-part-1/
- https://www.cnx-software.com/2017/10/16/esp32-micropython-tutorials/

- Any questions before you start this on your own?

John ‘Warthog9’ Hawley | warthog9@eaglescrag.net | @warty9