Discovering Tiny Snakes

IoT development without the need to compile (mostly)
Quick: MicroPython vs. CircuitPython?
Why is this different?
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
  - Low power MCUs and CPUs are getting more powerful, and cheaper at the same time.
- 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
How to get started

● Serial Drivers
  - Linux: Assuming your distro isn’t terrible, you are done

● Download / Install esptool
  - This requires Python
  - Linux: distro packages are available
  - Windows / Mac: use pypi to install

● Download MicroPython and Upload it to the board
  - esptool.py --chip esp32 --port /dev/ttyUSB0 erase_flash & & \ esptool.py --chip esp32 --port /dev/ttyUSB0 write_flash -z 0x1000 <path to micropython.bin>
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
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...

```
rst:0x1 (POWERON_RESET), boot:0x13 (SPI_FAST_FLASH_BOOT)
configsp: 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:0x40080000,len:5512
entry 0x40090114c
I (380) cpu_start: Pro cpu up.
I (380) cpu_start: Single core mode
I (380) heap_init: Initializing. RAM available for dynamic allocation:
I (392) heap_init: AT 3FFAE6E0 len 00001020 (6 Kib) : DRAM
I (396) heap_init: AT 3FFC4F48 len 0001B088 (108 Kib) : DRAM
I (405) heap_init: AT 3FFE0440 len 000038C0 (14 Kib) : D/IRAM
I (411) heap_init: AT 3FFE4350 len 0001BC00 (111 Kib) : D/IRAM
I (417) heap_init: AT 40091440 len 000006B8 (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-g105716514 on 2018-09-26; ESP32 module with ESP32
Type "help()" for more information.
```
Now to blink an LED!

- Type the following:

```python
from machine import Pin, Signal
import machine
pin_led_board = 0
_led_board = Pin(pin_led_board, Pin.OUT)
led_board = Signal(_led_board, invert=True)
led_board.off()
led_board.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](https://github.com/adafruit/ampy)

- Tab completion works in the repl prompt

- `<ctrl>`+`e` at the repl prompt puts you into “paste” mode, so you can paste a longer set of code into the buffer to execute
Let's get more advanced....

• Read from the BME280
  - Upload bme280.py to the board
  - Setup I2C in python
  - Attach bme280 to the I2C bus
  - Read some data
The Code:

```python
from machine import Pin, I2C
import machine
import bme280

pin_i2c_scl     = 22
pin_i2c_sda     = 21
bme280_address  = 0x77

print("Setting up Sensor I2C")

sensor_i2c = I2C( scl=Pin(pin_i2c_scl), sda=Pin(pin_i2c_sda) )

print("Setting up BME280")

bme = bme280.BME280( i2c=sensor_i2c, address=bme280_address )

bme.values
```
Adding the TSL2591 to the BME280

- Import the TSL2591 driver
- Attach the driver to the same I2C
The Code

```python
from machine import Pin, I2C
import machine
import bme280
import tsl2591

pin_i2c_scl   = 22
pin_i2c_sda   = 21

bme280_address = 0x77

print("Setting up Sensor I2C")

sensor_i2c = I2C( scl=Pin(pin_i2c_scl), sda=Pin(pin_i2c_sda) )

print("Setting up BME280")

bme = bme280.BME280( i2c=sensor_i2c, address=bme280_address )

bme.values

tsl = tsl2591.Tsl2591()
tsl.get_full_luminosity()
```
Where to go from here

• Setup Wifi in client mode
  - Then run:
    ```python
    import socket
    addr_info = socket.getaddrinfo("towel.blinkenlights.nl", 23)
    s = socket.socket()
    s.connect(addr)
    while True:
      data = s.recv(500)
      print(str(data, 'utf8'), end='')
    ```
    # the several enters above matter for the loop levels

• Setup Wifi in AP mode (note: it can do both simultaneously, albeit slowly)

• Install uMQTT and export sensors over MQTT

• Explore the “test” scripts included

• Put files on the sdcard

• Enjoy the board
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