MicroPython for the Internet of Things

Dr Glenn Ramsey

glenn@leaftech.nz

August 23, 2024

Introductions

Me, Dr Glenn Ramsey

- Professional: Software developer, Engineer (BE, ME, PhD) (Mechanical, Control Systems, Bioengineering, Mathematical analysis and modelling)
- Personal: Alternative physics, Motorbike trail riding

Helpers

- Here in the room:
  Rob van der Linde
  William Hamilton
- Workshop development and testing
  David Neil

Introductions

You and your fellow participants

- Find out about the people sitting next to you.
- Respect their communication preference traffic light
- Things you could ask:
  What is their experience with Python?
  Why are they at this workshop?

What is The *Internet of Things*?

Devices not requiring human interaction that are connected to the Internet.

Typically attached to sensors and “phoning home” with the sensor data. Data is typically displayed on a dashboard and is often stored in a time-series database.

In this workshop we will use a micro-controller device to collect data from the environment and report that data to a local server.

The micro-controller will be programmed using a variant of Python called MicroPython which is specialised for running on resource-limited hardware.

Introduction to MicroPython

What is MicroPython?

- A lean and efficient implementation of Python 3
- Designed to run on microcontrollers and in constrained environments
- Provides access to hardware-level functionalities

Why MicroPython (vs e.g. a C/C++ based toolchain)?

- It's Python - and therefore totally awesome!
- All of the under-the-hood details are taken care of.
- => Low barrier to entry.

https://docs.micropython.org

- Each device may have different features - there is device specific documentation

Differences to CPython

- There are many, but you probably won't notice.

e.g.

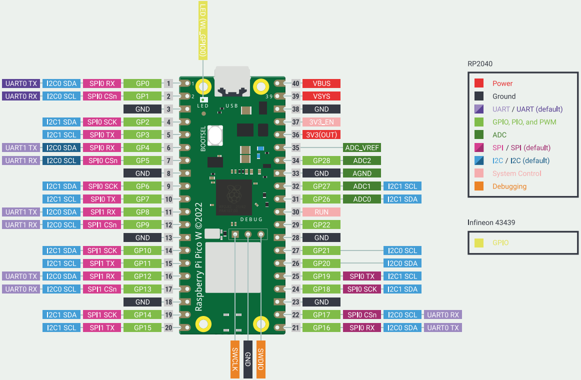
```
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
  ...
AttributeError: 'socket' object has no attribute '__exit__'
```

- Try it. Look up docs if it doesn't work. https://docs.micropython.org/en/latest/genrst/index.html#micropython-differences-from-cpython
- No source level debugger. i.e. no breakpoints

  Use `print(...)`, REPL, on-board LED

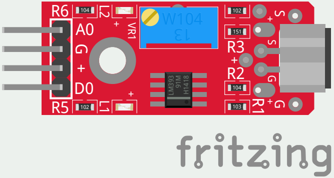
The hardware - Pico WH microcontroller

Beware: it is sensitive to static electricity



The hardware - KY-038 Sound sensor

KY-037 sound sensor



The one we have, KY-038, is similar but has a smaller microphone.

Task 1: Set up software on the host

You'll need at least:

- A text editor or IDE. Thonny is strongly recommended because of its integration with MicroPython and cross platform support.
- mpremote (optional)
  (in a venv is a common workflow)

```
# create a project dir, then
python -m venv <venv_dir>

. venv/bin/activate # Linux, MacOS

venv\Scripts\activate # Windows
pip install mpremote
```

Task 1 pg 2: Linux - serial port permissions

```
ls -l /dev/tty*
```

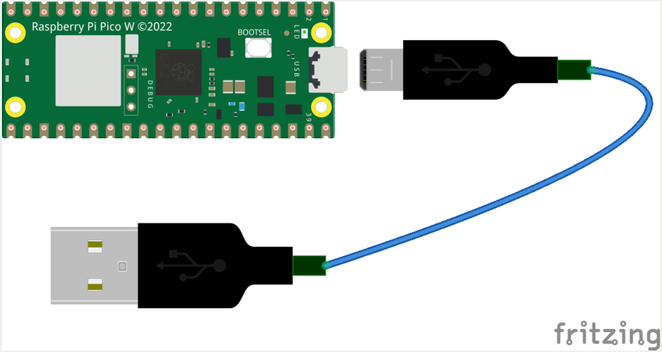
```
sudo adduser $USER --groups dialout # or the relevant group
```

then log out of desktop and log in again

Task 2: Set up the hardware - connect

Connect the Pico board to your computer

Beware: it is sensitive to static electricity



For safety (physical and electrical) leave the pins stuck into the foam packaging, for now.

Task 2: Set up the hardware - connect

- Connect the pico to your laptop using the USB cable.
- It should be in USB mass storage mode and your computer should detect it
- If not then disconnect then press and hold the BOOTSEL button while connecting the USB cable to your computer
- ***Do not*** connect the sound sensor at this stage.

Task 3: Set up the hardware - install MicroPython

Installing MicroPython on the Pico

Step 1. Download the .UF2 file from:
`https://www.raspberrypi.com/documentation/microcontrollers/micropython.html`

- Get the file from this link:
  Raspberry Pi Pico W with Wi-Fi and Bluetooth LE support

Step 2. Copy it to the device using your preferred method (drag'n'drop) or e.g.

```
cp ~/Downloads/RPI_PICO_W-20240602-v1.23.0.uf2
/run/media/glenn/RPI-RP2/
```

Task 4: Connect to the REPL

```
mpremote help  # to see what it can do
mpremote devs  # list connected devices
mpremote a0    # shortcut to connect device, c0 on Windows
               # see help
mpremote repl  # opens a Python prompt on the device

# Things to try from the REPL
  (quit Thonny if using a terminal)

>>> import machine
>>> dir(machine)
>>> help(machine)
>>> help(machine.Pin)
```

Also try `network`, `network.WLAN`, `bluetooth`

Task 5: Write and run the LED blinking program

Things you'll need:

- `docs.micropython.org`
- A text editor or IDE (Thonny recommended - set interp to CPython for now)

```
from machine import Pin
Pin("LED", Pin.Out)
utime.sleep_ms(...)
```

- A loop
- If using `mpremote` to copy the file to the board use:

  `mpremote fs cp blink.py :blink.py`
- Note the colon ":" before the destination filename
- `mpremote repl`
- `>>> import blink`

Task 5: A solution (others are possible)

```
from machine import Pin
import utime

led = Pin('LED', Pin.OUT)
# led = Pin('EXT_GPIO0', Pin.OUT) # also works
delay = 100
while True:
    led.value(1) # or led.on()
    utime.sleep_ms(delay)
    led.value(0) # or led.off()
    utime.sleep_ms(delay)
```

Task 6/1: Connect to Wifi

Things you'll need:

```
wlan = network.WLAN(network.STA_IF)
wlan.active(True)

ssid = 'rp2-pico'
password = 'kiwipycon'

# Must set IP before connect or we get undefined response code 2
wlan.ifconfig(['192.168.2.1xx', '255.255.255.0',\
                '192.168.2.1', '1.1.1.1',])

wlan.connect(ssid, password)

# Wait for connection
while wlan.status() != network.STAT_GOT_IP:
    utime.sleep_ms(1000)
```

Where 1xx is your assigned number (in range 101-150).

Task 6/2: Get data from the Internet

Use `urequests.get(...)` to obtain data from the following URLs:

- `http://ip.jsontest.com/`
- `http://date.jsontest.com/`

  I am recommending these URLs because the data size is small.
- It would be useful to write a function to do the WiFi connection so it can be reused.

Task 6/2: Get data from the Internet

```
url_list = ["http://ip.jsontest.com/", \
            "http://date.jsontest.com/"]

for url in url_list:

    print('about to get', url)

    # Get the data from the server
    response = urequests.get(url)

    # Print the server's response
    print(response.text)
```

Task 7: Send data to a server using 3 different methods

Use the board internal temperature as the data.

```
from machine import ADC
...
adc = ADC(ADC.CORE_TEMP) # ADC.CORE_TEMP == 4
                         # the internal temperature sensor
adc_voltage = adc.read_u16() * 3.3 / 65535
temp_c = round(27 - (adc_voltage - 0.706)/0.001721, 1)
```

- Send your name and the measured temperature as a single string.
- Bonus: Which method uses the least amount of memory? (use `gc.mem_free()`)

Task 7/1: Send data to a server using HTTP POST

1. Use HTTP POST to send a Python dict

```
data = { 'text': '<name> ' + str(temp_c),
         'color': '#RRGGBB' }
```

- Server is on http://192.168.2.2:8000
- Server uses your IP address as grid index.

```
response = urequests.post(url, json=data)
```

Task 7/2: Send data to a server using raw UDP

2. Use a raw UDP socket.

   Server is on http://192.168.2.2:8001

   Packet format is: (octets) `L S S ... R G B I I`

   (string length, string chars, color, 16 bit int MSB first)

Task 7/2: CPython UDP client example

Should be the same in MicroPython

```
import socket
udp_ip = "192.168.2.2"
udp_port = 8001

text = "Hello1234567890"
text_length = len(text)
color_rgb = (255, 0, 255)  # Magenta color
value = 12345
msb = (value >> 8) & 0xFF # most significant byte
lsb =  value & 0xFF       # least significant byte
message = bytearray([text_length]) + text.encode('utf-8') \
  + bytearray(color_rgb) + bytearray((msb, lsb))

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
n = sock.sendto(message, (udp_ip, udp_port))
print("sent", n)
```

Task 7/3: Send data to a server using MQTT

3. Use MQTT

```
# connect to Wifi
>>> import mip
>>> mip.install("umqtt.simple")
```

```
topic = "grid/<id>"
```

Server is on http://192.168.2.2

Same payload as for UDP.

See also: `struct.pack('>H', ...)`

Task 7/3: MQTT client partial example

```
from umqtt.simple import MQTTClient

... # connect to Wifi

# Define MQTT parameters
broker_address = "192.168.2.2"  # Use this IP address
client_id = "<your name>"+str(<your_id>)
topic = b"grid/<your_id>" # Your assigned ID

# < assemble message as for UDP client >

client = MQTTClient(client_id, broker_address)
client.connect()
client.publish(topic, message)
client.disconnect()
```

Task 8: Toggle the LED using the button

Things you'll need:

- rp2.bootsel_button() Reports the state of the on-board button.
- The LED Pin from previous task.
- A loop
- Compare current button state to previous state.
- It could be tricky to get the logic right.

Task 8: A solution

```
from rp2 import bootsel_button
from machine import Pin
import time

led =  Pin('LED', Pin.OUT)
oldstate = False
while True:
    newstate = bootsel_button()
    if newstate == True and oldstate == False:
        led.toggle()
    oldstate = newstate
```

Task 9: Change the grid colour depending on button press time

Set grid colour to green.

While the button is pressed increment a counter by 1 every 50ms.

When the counter is between 20 and 40 make the grid orange.

When the counter is above 40 make the grid red.

When the button is not pressed decrement the counter by 1 every 50ms.

Things you might need:

- `micropython.schedule()`
- `gc.collect() # I will explain`

Task 9: A solution for a count up/down loop

```
# imports, connect to WiFi, set up constants
count = 0
while True:
    if bootsel_button():
        count += 1
    else:
        count -= 1
        if count < 0:
            count = 0

    if count >= 40:
        micropython.schedule(set_colour, 'red')
    elif count >= 20:
        micropython.schedule(set_colour, 'orange')
    else:
        micropython.schedule(set_colour, 'green')

    time.sleep_ms(50)
```

Task 9: Sending the data

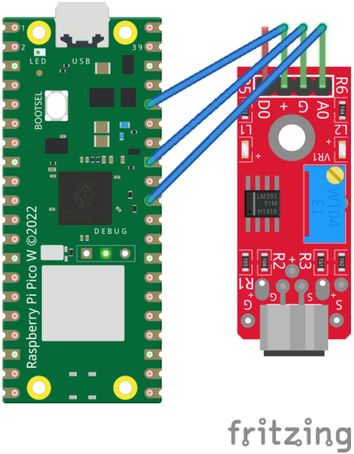
```
def set_colour(arg):
    global last_col
    if arg != last_col:
        print(arg)
        last_col = arg

        data = {
            'text': "Glenn",
            'color': arg # named or hex code
            }

        print('about to post to', url, 'with data', data)
        # Post the data to the server
        response = urequests.post(url, json=data)
    gc.collect() # because urequests.post(...) consumes memory
```

Task 10: Connect sound sensor

Disconnect USB first!



36 — + , 33 — G , 31 — A0

Task 11: Read sound level from the sound sensor continuously

Things you'll need:

- `ADC(0) # or ADC(Pin(26))`
- A loop
- Use `print(...)` to display the value

Task 11: A solution

```
# Set up ADC to read voltage from sound sensor
adc = ADC(0) # or ADC(Pin(26))

while True:
    sample = adc.read_u16()
    print(sample)
```

- What do you notice about the data?
- Why is it like this?
- How do we “fix” it?

Task 12: Measure the loudest sound in a 50 ms period

Calculate sound amplitude using difference of maximum and minimum values.

Things you'll need:

- A loop
- Sound samples i.e. `sample = adc.read_u16()`
- Time measurement
  - utime.ticks_ms()
  - utime.ticks_diff()

Task 12: A solution

```
adc = ADC(0) # or ADC(Pin(26))

while True:
    start = utime.ticks_ms()
    sample_max = 0 # minimum possible sample value
    sample_min = 2**16 # maximum possible sample value
    while utime.ticks_diff(utime.ticks_ms(), start) < 50:
        sample = adc.read_u16()
        if sample > sample_max:
            sample_max = sample
        elif sample < sample_min:
            sample_min = sample
    level = sample_max - sample_min
    print(level)
```

- Why do we need `utime.ticks_diff(...)`?

Task 13: Sound meter program - requirements

The sound meter responds to sound level and produces output:

- green - sound level acceptable
- orange - sound level tolerable
- red - sound level unacceptable

After measuring a level greater than green the output decays over time back through the lower levels.

I.e. if red is measured then it stays red for a few seconds, then goes orange for a few seconds then goes back to green.

Initially use print("red" | "orange" | "green") to display the output.

Once that is working add code to send the data to the "grid" server.

Task 13: Sound meter program - hints

Hints:

- Combine code patterns from task 9 (sending data) and task 12 (sound level measurement)
- Use peak to peak sound samples
- Send the data by HTTP POST, UDP, or MQTT
- The POST method understands named web colours. i.e. “red”, “orange”, or “#00FF00”
- red: `(255, 0, 0)`, orange: `(255, 255, 0)`, green: `(0, 255, 0)`

Task 14/1: Blink the LED precisely

Interrupts

- Phone call analogy
- Short duration
- Memory allocation rules

Things you'll need:

- `machine.Timer`
- Interrupt handler

Also of note:

- `micropython.schedule` – search on `docs.micropython.org`

Task 14/2: A solution

```
from machine import Pin, Timer

led = Pin("LED", Pin.OUT)
tim = Timer()

def tick(timer):
    global led
    led.toggle()

tim.init(freq=2.5, mode=Timer.PERIODIC, callback=tick)

while True:
    print("waiting around, doing nothing")
    utime.sleep(2)
```

- See the live example

Task 15: Change the blink frequency on button press

Things you'll need:

- rp2.bootsel_button()
  Reports the state of the on-board button.

Task 16/1: Create a reaction timer game using the onboard button and LED

- Measure the elapsed time from illuminating the LED until the button is pressed
- You will need to poll the button

  Ideally a GPIO interrupt would be used but this is not possible with the onboard button
- There probably needs to be a "get ready" signal
- The time between "get ready" and start of timing needs to be slightly random to prevent anticipation. (`urandom.uniform`)
- Report the reaction time to the REPL using `print(...)`

Task 16/2: (optional) Estimate the probable error range for the reaction time

- Method is up to you - see what you can come up with.
- I have some ideas.

Task 17: Report the reaction time to the leaderboard server

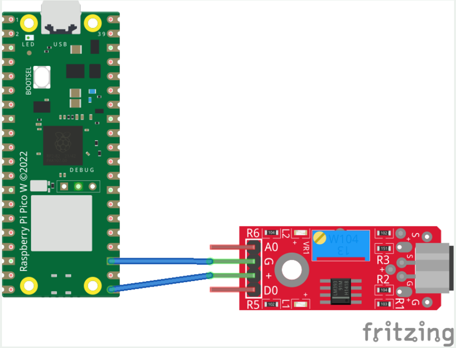
Things you'll need: (c.f. task 7/3 MQTT)

```
broker_address = "192.168.2.2"
client_id = "<your name>"+str(<your_id>)
topic = b"reaction"
client = MQTTClient(client_id, broker_address)
...
data = {
  'name': '<your name>',
  'score': <reaction_time>
}
client.connect()
client.publish(topic, json.dumps(data))
client.disconnect()
```

Task 18: Connect the sound sensor to use its LEDs

"Off label" usage (current draw OK for the GPIO pins)

Disconnect USB first!



23(G) — G, 21 (GP16) — +

Task 19/1: Make the LEDs "breathe" using pulse width modulation

Things you'll need:

```
machine.Pin
machine.Timer
machine.PWM

pwm.duty_u16(...)
```

- A loop
- Timing

Task 19/2: A solution (pg 1)

```
from machine import Pin
from machine import Timer
from machine import PWM
from time import sleep

led = Pin(16, Pin.OUT)
pwm = PWM(led)
duty_step = 129  # Step size for changing the duty cycle
freq = 5000
pwm.freq(freq)
```

Task 19/3: A solution (pg 2)

```
try:
    while True:
      # Increase the duty cycle gradually
      for duty_cycle in range(8*duty_step, 65536, duty_step):
        pwm.duty_u16(duty_cycle)
        sleep(0.005)
      # Decrease the duty cycle gradually
      for duty_cycle in range(65536, 8*duty_step, -duty_step):
        pwm.duty_u16(duty_cycle)
        sleep(0.005)
      sleep(1.5)
except KeyboardInterrupt:
    print("Keyboard interrupt")
    pwm.duty_u16(0)
    print(pwm)
    pwm.deinit()
```

Task 20: Report the sound level via a local web server

- Implement a local webserver. I recommend Microdot.
- Use the Pico as a Wifi AP and connect from your phone or laptop
- Or use as a station (client as in previous tasks) and get your neighbour to `urequests.get()` from your device
- Display the data using HTML (or just plain text)
- Many examples online.

  https://microdot.readthedocs.io/en/latest/intro.html#micropython-installation

Task 20 pg 2: Things you might need:

AP mode

```
ap = network.WLAN(network.AP_IF)
ap.config(essid=ssid, password=password)
ap.active(True) # Default IP is 192.168.4.1
                # It has a DHCP server
while ap.active() == False:
    pass
```

Http server

```
from microdot import Microdot
app = Microdot() # default port is 5000
@app.route('/')
async def index(request):
    return 'MicroPython is awesome!'
app.run()
```

Task 21: Bluetooth low energy central and peripheral

- Work in pairs (or groups)
- On one device create a BLE peripheral to publish temperature
- On the other device create a BLE central to read the temperature and send to the server over Wifi

Things you'll need:

```
https://github.com/micropython/micropython-lib
/tree/master/micropython/bluetooth/aioble/examples/

mip.install("aioble")

temp_client.py

temp_sensor.py
```

Extra projects

- The Arduino sensor kit
- 6 dof accelerometer
- I2C Wattmeter
- Laser ToF sensor
- Light beam sensor
- Humidity sensor
- Rotary encoder