

# CSCI 4907

# Introduction to IoT and Edge Computing Applications

Prof. Kartik Bulusu, CS Dept.

Week 4 [02/16/2024]

- Guest lecture: **Anatomy of a real life IoT attack - Side Channel Attack (SCA) Analysis** by Chowdary Yanamadala, Technology Strategist ARM
- Discussion on the midterm projects for updates and reporting
- dweet.io – Twitter of things
- Fetching data from dweet
- dweet an LED - Your first IoT program Raspberry Pi messaging with Python scripts
- HW 3 assigned

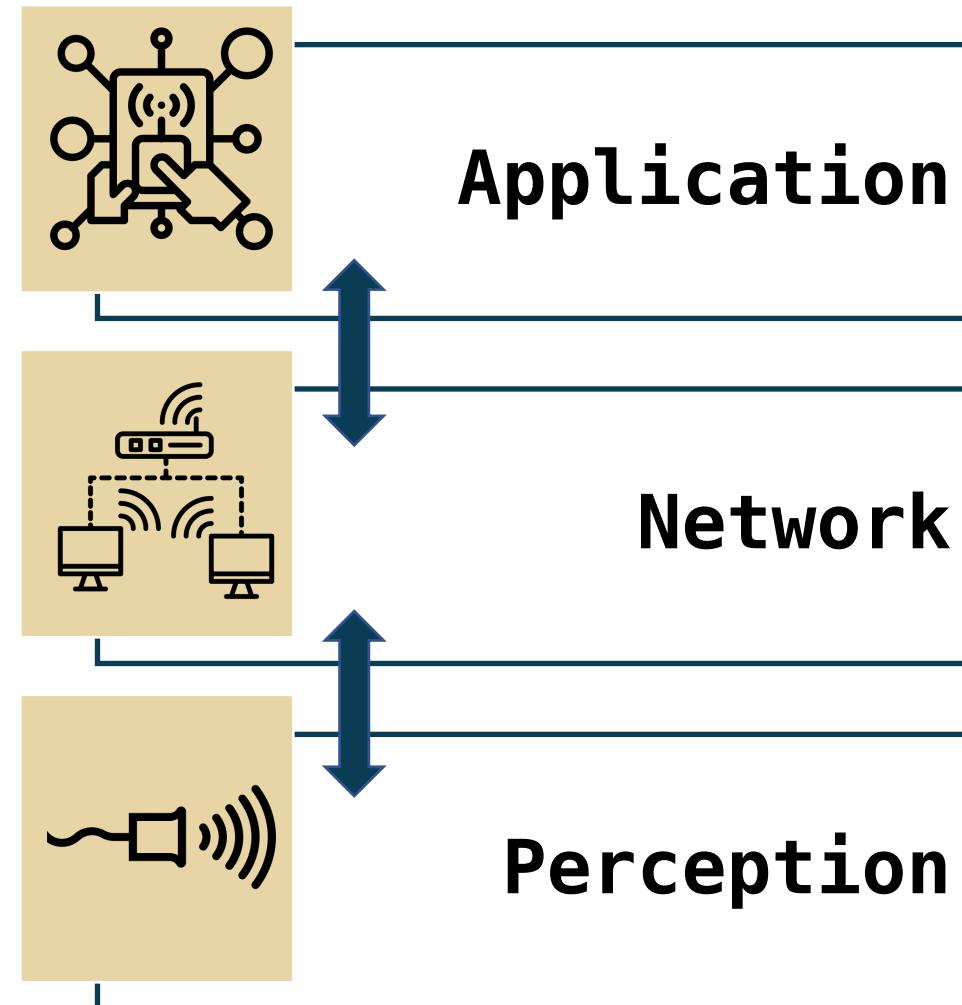
```
git clone git@github.com:gwu-csci3907/Spring2024.git
```

```
git clone https://github.com/gwu-csci3907/Spring2024.git
```

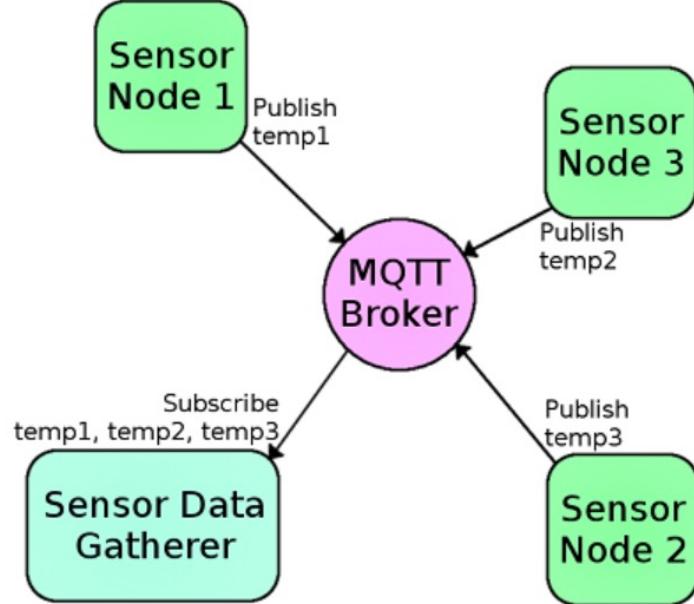


School of Engineering  
& Applied Science

## Recap: The 3-Layer IoT Architecture



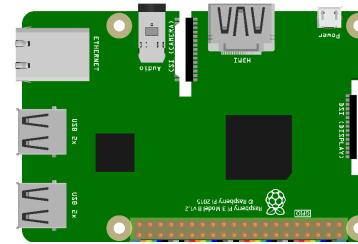
# MQTT paradigm



## Hardware

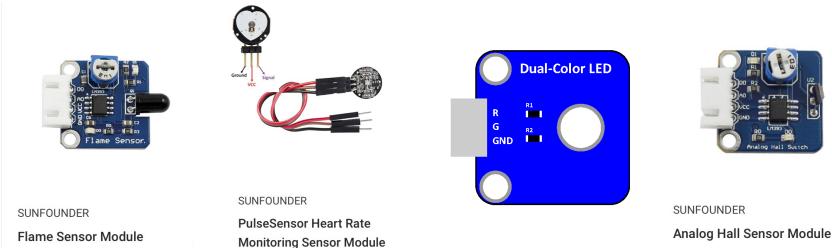
### Broker

- The broker is the server
- It distributes the information to the interested devices connected to the server.



### Client

- The device that connects to broker to send or receive information.



## Messaging

### Topic

- The name that the message is about.
- Clients publish, subscribe, or do both to a topic.

### Publish

- Clients that send information to the broker to distribute to interested clients based on the topic name.

### Subscribe

- Clients tell the broker which topic(s) they're interested in.

### QoS

- Quality of Service to the broker
- Integer value ranging from. 0-2.



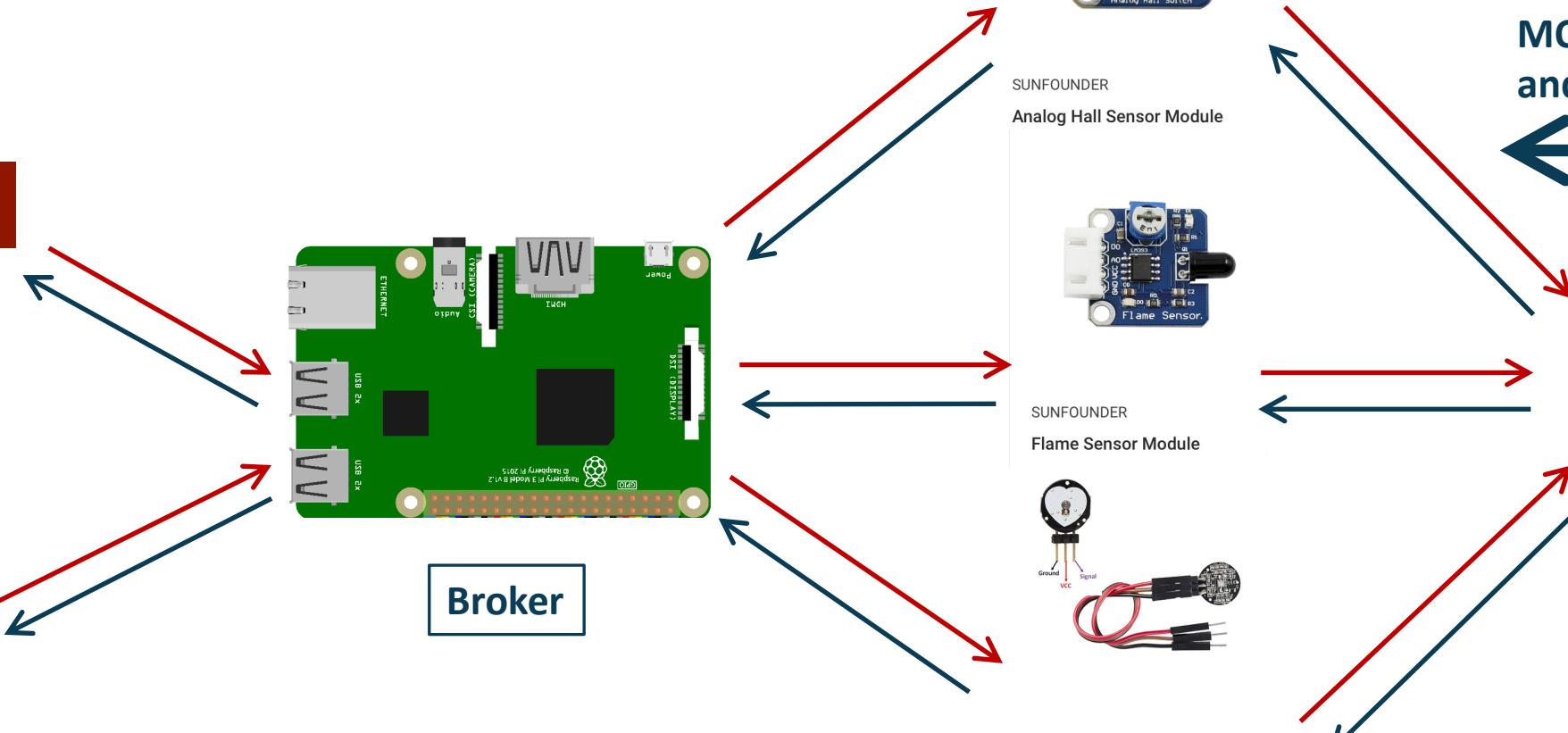
# Practical view of MQTT in IoT applications

Node-RED

dweet.io

Clients

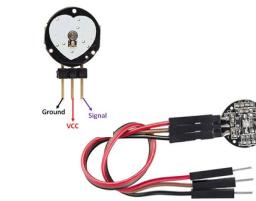
Broker



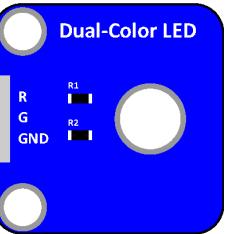
SUNFOUNDER  
Analog Hall Sensor Module



SUNFOUNDER  
Flame Sensor Module



SUNFOUNDER  
PulseSensor Heart Rate  
Monitoring Sensor Module



Client

MQTT to control  
data output

MQTT to read  
and publish data

dweet.io  
A simple publishing and subscribing tool  
for IoT devices ..

or anybody's thing

It's like Twitter of things!

# Goal-1:

## Fetch dweet-data using Python

### Step-1:

#### Install Thonny IDE

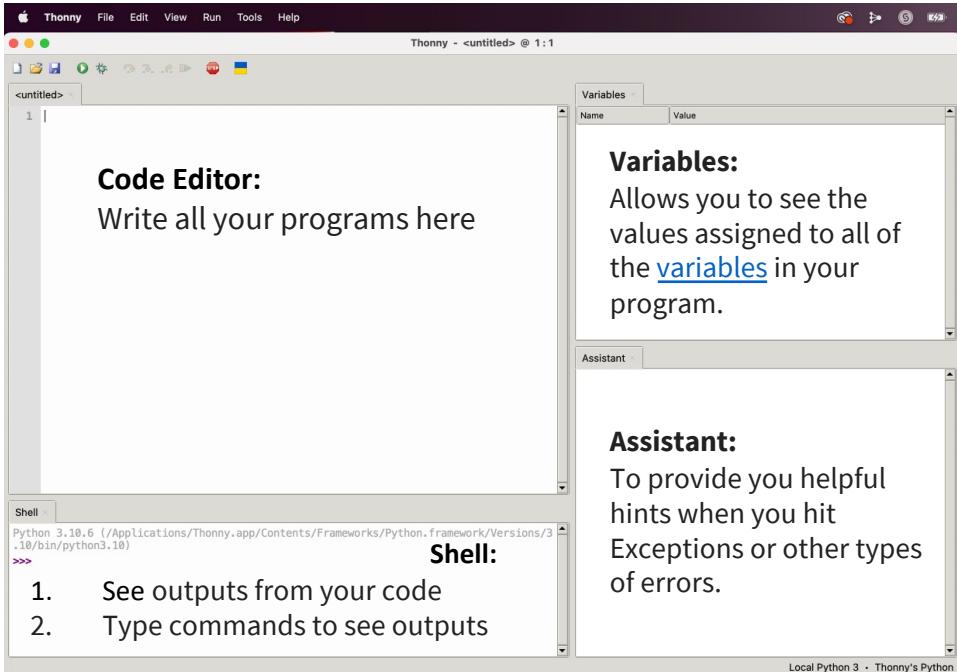
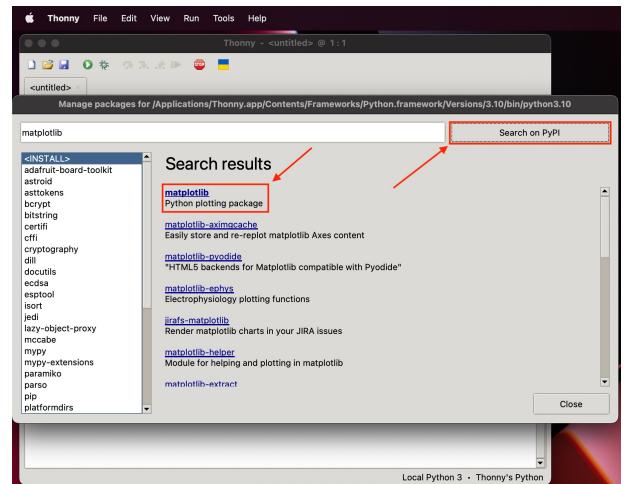
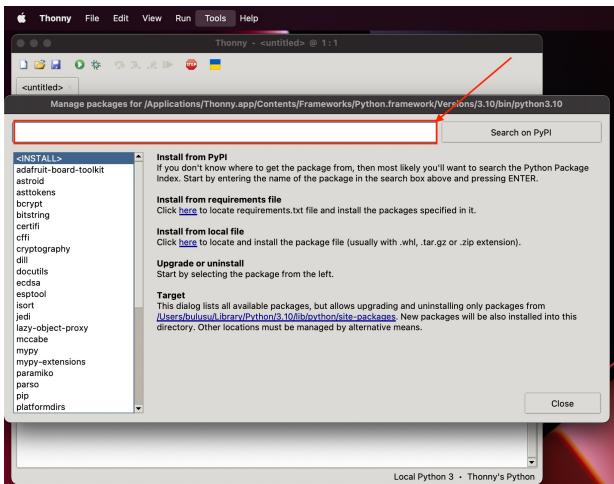
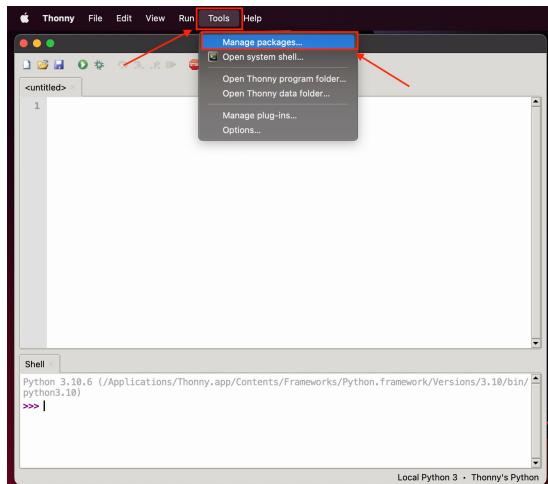


<https://thonny.org>

### Step-2

#### Install the following libraries:

1. matplotlib
2. pandas
3. numpy
4. dweepy



## Step-3: Go to dweet.io



<https://dweet.io>

Ridiculously simple messaging  
for the Internet of Things.

Fast, free and ridiculously simple— it's like Twitter  
for social machines.

If your product, device, machine, gadget or thing can connect to the Internet, it  
can use dweet.io to easily publish and subscribe to data.

dweet.io doesn't require any setup or sign-up— just publish and go. It's  
machine-to-machine (M2M) for the Internet Of Things (IOT) the way it was  
meant to be.

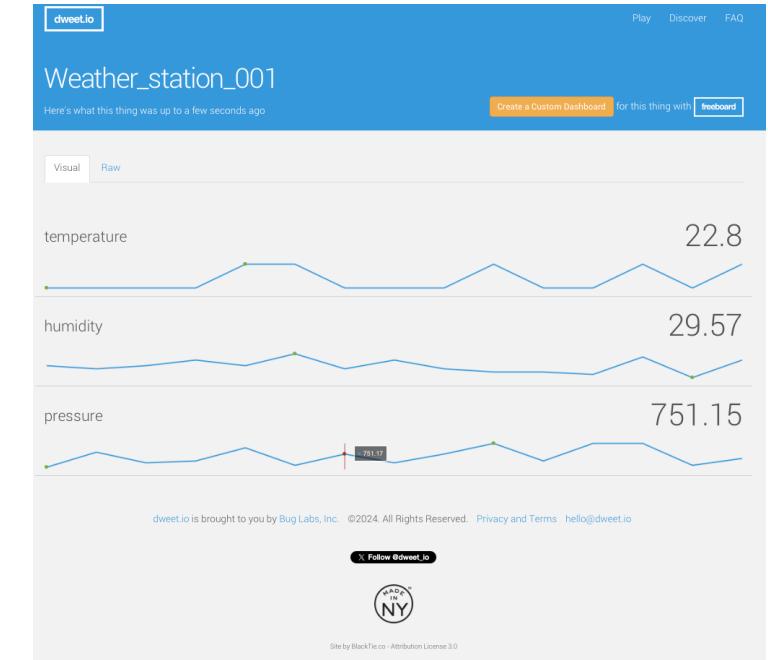


Ridiculously simple messaging  
for the Internet of Things.  
Fast, free and ridiculously simple— it's like Twitter  
for social machines.

If your product, device, machine, gadget or thing can connect to the Internet, it  
can use dweet.io to easily publish and subscribe to data.

dweet.io doesn't require any setup or sign-up— just publish and go. It's  
machine-to-machine (M2M) for the Internet Of Things (IOT) the way it was  
meant to be.

dweet.io		
		Play Discover FAQ
00001d9ad1f6	uptime, appUptime, memory, cpuT	a few seconds ago
00001996614e	uptime, appUptime, memory, cpuT	a few seconds ago
00001ea7c70c	uptime, appUptime, memory, cpuT	a few seconds ago
00001ea77fba	uptime, appUptime, memory, cpuT	a few seconds ago
Branon-the-hole-wd	heartbeat, Timestamp	a few seconds ago
00001d9b438a	uptime, appUptime, memory, cpuT	a few seconds ago
00001d9b24e7	uptime, appUptime, memory, cpuT	a few seconds ago
Weather_station_001	temperature, humidity, pressure	a few seconds ago
00001d9b6669	uptime, appUptime, memory, cpuT	a few seconds ago
00001cd42e82	uptime, appUptime, memory, cpuT	a few seconds ago
00001d9ab254	uptime, appUptime, memory, cpuT	a few seconds ago
00001ea71093	uptime, appUptime, memory, cpuT	a few seconds ago
00001b2ec336	uptime, appUptime, memory, cpuT	a few seconds ago
00001cd43023	uptime, appUptime, memory, cpuT	a few seconds ago
00001ea75c9f	uptime, appUptime, memory, cpuT	a few seconds ago



# Step-4:

## Let's write a Python program to retrieve dweet-data from a thing

## Test drive:

<http://www.dweet.io/follow/Weather station 001>

## dweet-thing: Weather\_station\_001

**data1: temperature**

## data2: humidity

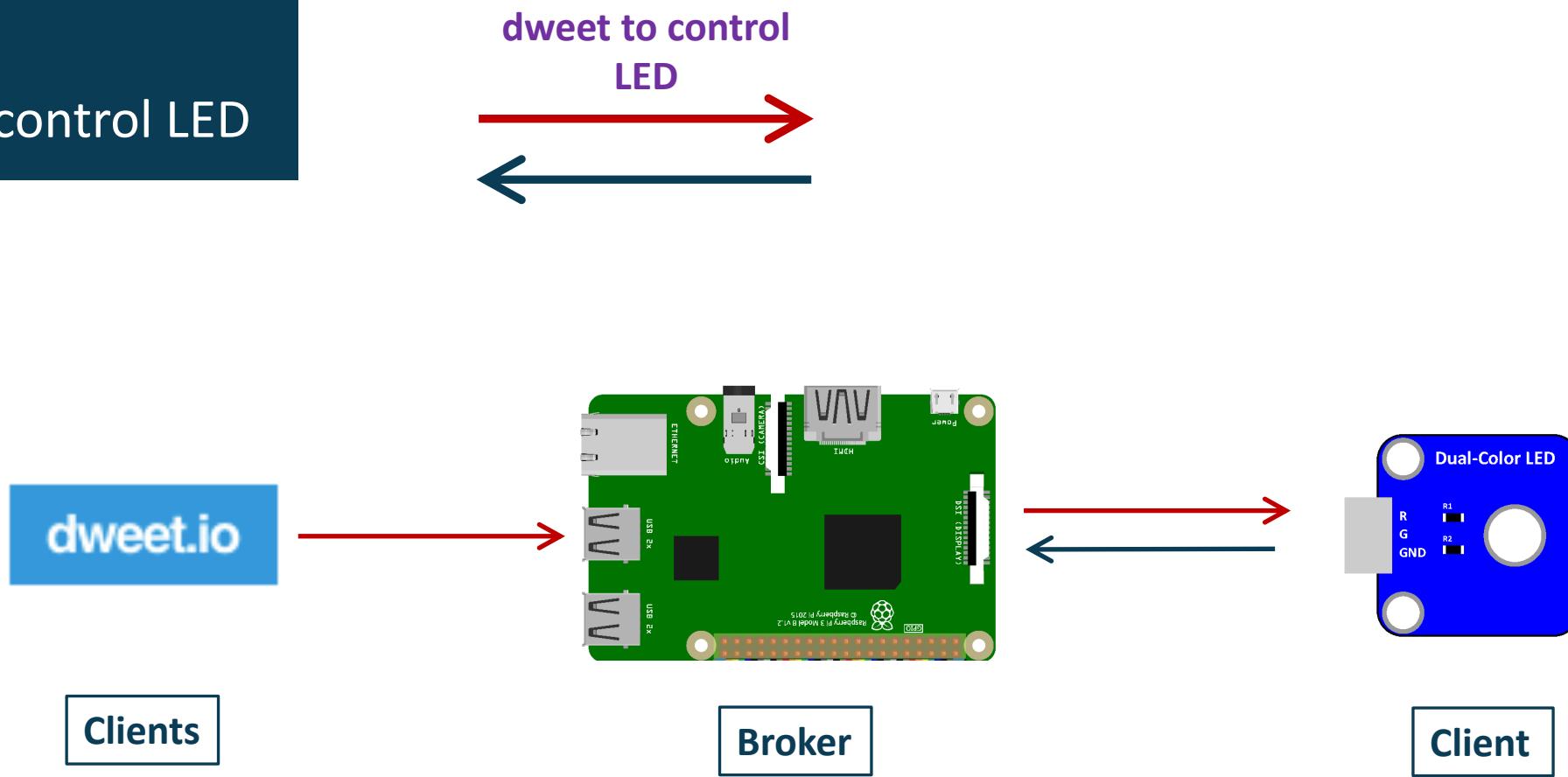
## **data3: pressure**

The screenshot shows the Thonny Python IDE interface with the following components:

- Code Editor:** The main window displays the code for `dweetRetrieve.py`. The code imports `dweeypy`, gets a thing name from input, and retrieves IoT data for three different metrics (Temperature, Humidity, Barometric Pressure) from a thing named "4C\_Seattle1". It also prints the status update and date/time.
- Variables View:** A sidebar panel titled "Variables" lists the current state of variables. It includes:
  - `data1`: 'Temperature'
  - `data2`: 'Humidity'
  - `data3`: 'Barometric\_Pressure'
  - `dateNow`: '2023-02-24'
  - `date_created`: '2023-02-24T02:04:30.933Z'
  - `dweeypy`: <module 'dweeypy' from '/Users/bulusu/Li28'
  - `humidity`: 28
  - `pressure`: 1005.25
  - `stampTime`: '02:04:30'
  - `tempC`: 66.02
  - `tempF`: 3801.6
  - `thing`: '4C\_Seattle1'
  - `thing_info`: [{"thing": "4C\_Seattle1", "created": "2023-02-24T02:04:30.933Z"}]
  - `this_dict`: None
- Assistant View:** A sidebar panel titled "Assistant" provides feedback on the code:
  - The message: "The code in [dweetRetrieve.py](#) looks good."
  - A note: "If it is not working as it should, then consider using some general [debugging techniques](#).
  - A link: "[Was it helpful or confusing?](#)"
- Shell View:** The bottom panel shows the output of running the script:

```
>>> %Run dweetRetrieve.py
Enter the thing name: 4C_Seattle1
Retrieve IoT data #1 from the thing: Temperature
Retrieve IoT data #2 from the thing: Humidity
Retrieve IoT data #3 from the thing: Barometric_Pressure
Status update for 4C_Seattle1
Date 2023-02-24
Time 02:04:30
```

## Goal-2: dweet to control LED



# Let's get started

## Preliminaries:

### 1. git-clone

z

```
git clone git@github.com:gwu-i3907/Spring2024.git
```

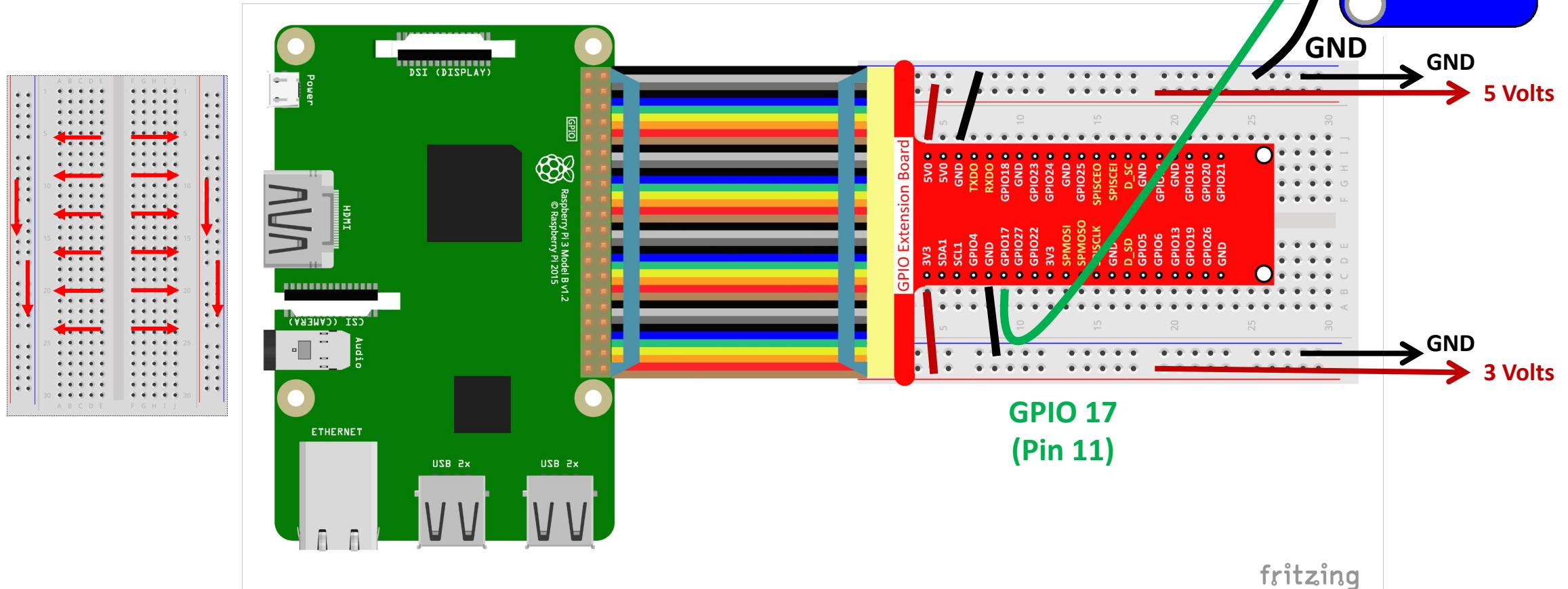
```
git clone https://github.com/gwu-csci3907/Spring2024.git
```

z

### 2. I have the files on a USB in case there is a slowdown!

## Step-0:

Put together the LED circuit connections on your Raspberry Pi 4B



Run  
dweet\_LED\_Example

Your first IoT program

Graded Lab Activity (10 points)

## Step-1:

- Create **python** virtual environment
  - Activate **python-venv**
  - Install **pip**

```
pi@raspberrypi:~$ cd Desktop/Spring2024/dweet_LED_Example/  
pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $  
pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ python -m venv venv  
pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ source /venv/bin/activate  
(venv) pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ pip install --upgrade pip  
(venv) pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ pip list
```

## Step-2:

- Review of the codes provided to you
- Execute `gpio_pkg_check.py` from the terminal
- See example screenshot for Thonny IDE

Note: You are still in `python-venv`

The screenshot shows the Thonny IDE interface. The code editor window displays the `gpio_pkg_check.py` script. The script checks for the availability of Python GPIO libraries: `GPIOZero`, `PiGPIO`, and `RPi.GPIO`. It prints 'Available' if import succeeds or 'Unavailable. Install with "pip install [library]"' if it fails. The script is run in the Thonny shell, which shows the Python version and a prompt. The right side of the interface includes a Variables and Assistant panel.

```
#!/usr/bin/env python3
# Author: Kartik Bulusu
# Modified by Kartik Bulusu (CS Dept., GWU)
#
# This Python script checks for the availability of various Python GPIO Library Packages.
# It does this by attempting to import the Python package. If the package import is successful,
# we report the package as Available, and if the import (or import initialization) fails,
# we report the package as Unavailable.
#
# Dependencies:
#   pip3 install gpiozero pigpio RPi.GPIO
#
# Built and tested with Python 3.7 on Raspberry Pi 4 Model B
# Tested with Python 3.5.3 on Raspberry Pi 3B+
"""
try:
    import gpiozero
    print('GPIOZero Available')
except:
    print('GPIOZero Unavailable. Install with "pip install gpiozero"')

try:
    import pigpio
    print('PiGPIO Available')
except:
    print('PiGPIO Unavailable. Install with "pip install pigpio"')

try:
    import RPi.GPIO
    print('RPi.GPIO Available')
except:
    print('RPi.GPIO Unavailable. Install with "pip install RPi.GPIO"')
```

```
(venv) pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ python gpio_pkg_check.py
```

## Step-3: Install GPIO & requests packages

### Note:

- You are still in python-venv
- Installation of all the packages may take a while
- You may need to repeat this step again.
  - Follow the subsequent steps to complete the installations

```
(venv) pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ pip install gpiozero pigpio  
RPi.GPIO numpy matplotlib requests yagmail
```

```
pi@raspberrypi:~/Desktop/Spring2023/LED_examples  
File Edit Tabs Help  
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ python gpio_pkg_check.py  
GPIOZero Unavailable. Install with "pip install gpiozero"  
PigPIO Unavailable. Install with "pip install pigpio"  
RPi.GPIO Unavailable. Install with "pip install RPi.GPIO"  
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $  
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $  
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ pip install gpiozero pigpio RPi.GPIO  
DEPRECATION: Python 3.5 reached the end of its life on September 13th, 2020. Please upgrade your Python as Python 3.5 is no longer maintained. pip 21.0 will drop support for Python 3.5 in January 2021. pip 21.0 will remove support for this functionality.  
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple  
Collecting gpiozero  
  Using cached https://www.piwheels.org/simple/gpiozero/gpiozero-1.6.2-py2.py3-none-any.whl (148 kB)  
Collecting pigpio  
  Using cached https://www.piwheels.org/simple/pigpio/pigpio-1.78-py2.py3-none-any.whl (39 kB)  
Collecting RPi.GPIO  
  Using cached https://www.piwheels.org/simple/rpi-gpio/RPi.GPIO-0.7.1-cp35-cp35m-linux_armv7l.whl (68 kB)  
Collecting colorzero  
  Using cached https://www.piwheels.org/simple/colorzero/colorzero-2.0-py2.py3-none-any.whl (26 kB)  
Requirement already satisfied: setuptools in ./venv/lib/python3.5/site-packages (from colorzero->gpiozero) (33.1.1)  
Installing collected packages: colorzero, RPi.GPIO, pigpio, gpiozero  
Successfully installed RPi.GPIO-0.7.1 colorzero-2.0 gpiozero-1.6.2 pigpio-1.78  
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
```

## Step-4:

Install system level dependencies needed to work with the following packages:

- [matplotlib](#)
- [numpy](#)
- [yagmail](#)

Note:

- These updates may take a while to complete
- Repeat installation [GPIO](#) & [requests](#) packages as suggested in Step-2 to complete all installations

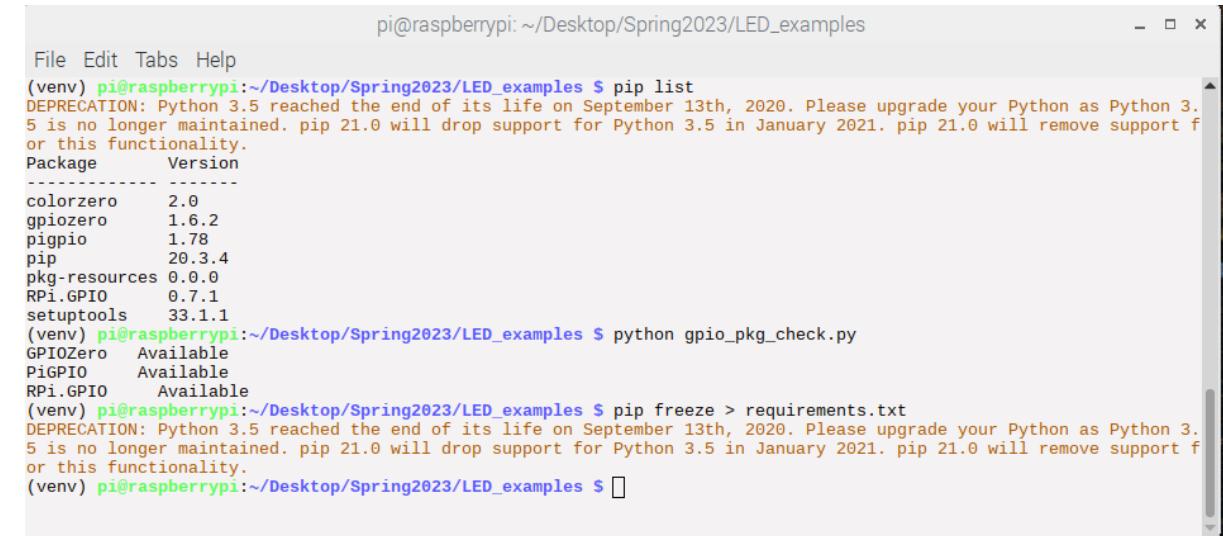
```
(venv) pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ sudo apt-get install libopenblas-dev
```

```
(venv) pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ sudo apt-get install libxml2-dev libxslt-dev
```

```
(venv) pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ pip install gpiozero pigpio RPi.GPIO numpy matplotlib requests yagmail
```

## Step-5:

- Execute `gpio_pkg_check.py` from the terminal
- Perform some checks and balances
  - `pip list`
  - `pip freeze`
  - See example screenshot for what to expect



The screenshot shows a terminal window titled "pi@raspberrypi: ~/Desktop/Spring2023/LED\_examples". It displays the following command and its output:

```
File Edit Tabs Help  
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ pip list  
DEPRECATION: Python 3.5 reached the end of its life on September 13th, 2020. Please upgrade your Python as Python 3.5 is no longer maintained. pip 21.0 will drop support for Python 3.5 in January 2021. pip 21.0 will remove support for this functionality.  
Package           Version  
----  
colorzero        2.0  
gpiozero         1.6.2  
pigpio          1.78  
pip              20.3.4  
pkg-resources    0.0.0  
RPi.GPIO         0.7.1  
setuptools       33.1.1  
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ python gpio_pkg_check.py  
GPIOZero Available  
PiGPIO   Available  
RPi.GPIO Available  
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ pip freeze > requirements.txt  
DEPRECATION: Python 3.5 reached the end of its life on September 13th, 2020. Please upgrade your Python as Python 3.5 is no longer maintained. pip 21.0 will drop support for Python 3.5 in January 2021. pip 21.0 will remove support for this functionality.  
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
```

(venv) pi@raspberrypi:~\$ /Desktop/Spring2024/dweet\_LED\_Example \$ python gpio\_pkg\_check.py

(venv) pi@raspberrypi:~\$ /Desktop/Spring2024/dweet\_LED\_Example \$ pip list

(venv) pi@raspberrypi:~\$ /Desktop/Spring2024/dweet\_LED\_Example \$ pip freeze > requirements.txt

## Step-6: Initialize **pigpio** daemon

```
(venv) pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ sudo pigpiod
```

**Alternatively:**

Initialize **pigpio** daemon using the following terminal commands

```
(venv) pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ sudo systemctl enable pigpiod
```

```
(venv) pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ sudo systemctl start pigpiod
```

```
(venv) pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ sudo systemctl stop pigpiod
```

```
(venv) pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ sudo systemctl start pigpiod
```

## Step-7: Run the **python** codes related to **dweet** following the next set of steps

## Breakdown of your first IoT program

## Skeleton of the updated Python program written for Raspberry Pi

```
import library1 as name1
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BORD)
GPIO.setup(12, GPIO.OUT)

def function_name(arguments):
    statement
    statement
    ...
    return value

if __name__ == "__main__":
    try:
        function_name1()
    except KeyboardInterrupt:
        function_name2()
```

The diagram illustrates the structure of a Python program for a Raspberry Pi. It uses curly braces and dashed arrows to map specific code snippets to their intended purpose. The annotations are as follows:

- Imports:** The first three lines of code (`import library1 as name1`, `import RPi.GPIO as GPIO`, and `import time`) are grouped by a brace on the left and an arrow pointing to the right, which points to the text: "Import libraries that are relevant for interaction with the Raspberry Pi hardware such as GPIO pins, camera ports etc."
- GPIO Configuration:** The two lines of code (`GPIO.setmode(GPIO.BORD)` and `GPIO.setup(12, GPIO.OUT)`) are grouped by a brace on the left and an arrow pointing to the right, which points to the text: "Set up GPIO pins as data outputs or inputs for sensors and actuators"
- User-defined Functions:** The block starting with `def function_name(arguments):` and ending with `return value` is grouped by a large brace on the left and an arrow pointing to the right, which points to the text: "Create user-defined functions to modularize your code and make it easy to work." Below this, another brace on the left and an arrow pointing to the right, which points to the text: "Create user-defined functions to release the GPIO pins"
- Entry Point and Exception Handling:** The block starting with `if __name__ == "__main__":` and ending with `function_name2()` is grouped by a large brace on the left and an arrow pointing to the right, which points to the text: "Create entry point into the program and pull all functions in". Below this, another brace on the left and an arrow pointing to the right, which points to the text: "Create a keyboard-interrupt exception clause"



**signal module**, facilitates means through which a program can receive information (events from the Operating System and pass them to programs.

- For example, signal **SIGINT** is generated when we press the keystrokes Ctrl + C on our keyboard

**json** (JavaScript Object Notation) **module** used for storing and transporting data from a server to a web page.

**os module** in Python provides functions for interacting with the operating system such as

- Handling and creating directories
- Listing out Files and Directories
- Deleting Directory or Files using Python

```
import signal
import json
import os
import sys
import logging <-----|
from gpiozero import Device, LED
from gpiozero.pins.pigpio import PiGPIOFactory
from time import sleep
from uuid import uuid1 <-----|
import requests <-----|
import RPi.GPIO as GPIO
```

**requests module** is used making HTTP requests to a specified URL by sending and receiving data from websites by providing a uniform interface for both **get** and **post** methods.

For example,

- get method** is used to retrieve information from the given server using a given URL
- post method** sends the encoded user information appended to the page request.
- The page and the encoded information are separated by the '?' character.

**logging module**, tracks (disruptive) events that occur in a computer system, such as problems, errors or just information on current operations

**uuid** (Universally Unique Identifier) **module** in Python, is a 128-character string of alphanumeric variable type, that uniquely identifies an object, entity, or resource in both space and time of a table.

- For example, **uuid.uuid1()** creates a UUID by utilizing the computer's MAC address and the current time in accordance with the RFC 4122 definition.

Sources:

<https://docs.python.org/3/library/logging.html>

<https://docs.python.org/3/library/uuid.html>

<https://favtutor.com/blogs/uuid-python>

<https://www.geeksforgeeks.org/python-requests-tutorial/>

```
# Global Variables
LED_GPIO_PIN = 17 # GPIO Pin that LED is connected to
THING_NAME_FILE = 'thing_name.txt' # The name of our "thing" is persisted into this file
URL = 'https://dweet.io' # Dweet.io service API
last_led_state = None # Current state of LED ("on", "off", "blinking")
thing_name = None # Thing name (as persisted in THING_NAME_FILE)
led = None # GPIOZero LED instance
```

Review your LED connection to make sure you are connected to GPIO17.  
Got back to Step-0 to confirm.

```
# Initialize Logging
logging.basicConfig(level=logging.WARNING) # Global logging configuration
logger = logging.getLogger('main') # Logger for this module
logger.setLevel(logging.INFO) # Debugging for this file.
```

# (2) → **logging module**, tracks (disruptive) events that occur in a computer system, such as problems, errors or just information on current operations

```
# Initialize GPIO
Device.pin_factory = PiGPIOFactory()
```

→ **gpiozero module** is a simple interface to GPIO devices with [Raspberry Pi](#), developed and maintained by [Ben Nuttall](#) and [Dave Jones](#).

- It is a simpler alternative to [RPi.GPIO module](#)
- **Documentation:** <https://gpiozero.readthedocs.io/en/latest/>

```

# Function Definitions
def init_led():
    """Create and initialise an LED Object"""
    global led
    led = LED(LED_GPIO_PIN)
    led.off()

def resolve_thing_name(thing_file):
    """Get existing, or create a new thing name"""
    if os.path.exists(thing_file):
        with open(thing_file, 'r') as file_handle:
            name = file_handle.read()
            logger.info('Thing name ' + name + ' loaded from ' + thing_file)
            return name.strip()
    else:
        name = str(uuid1())[:8] # UUID object to string.
        logger.info('Created new thing name ' + name)

        with open(thing_file, 'w') as f:
            f.write(name)

    return name

```

} → **gpizero module**, is used to initialize the GPIO pin where the LED is connected.

# (3) → **uuid** (Universally Unique Identifier) **module** in Python creates your unique thing-name:

- **uuid.uuid1()** creates a UUID by utilizing the computer's MAC address and the current time in accordance with the RFC 4122 definition.

# (4) → **logging module**, tracks (disruptive) events that occur in a computer system:

- **logging.info()** records information on a piece of code that works as intended.

# (5) →

```

def get_latest_dweet():
    """Get the last dweet made by our thing."""
    resource = URL + '/get/latest/dweet/for/' + thing_name
    logger.debug('Getting last dweet from url %s', resource)

    r = requests.get(resource)

    if r.status_code == 200:
        dweet = r.json() # return a Python dict.
        logger.debug('Last dweet for thing was %s', dweet)

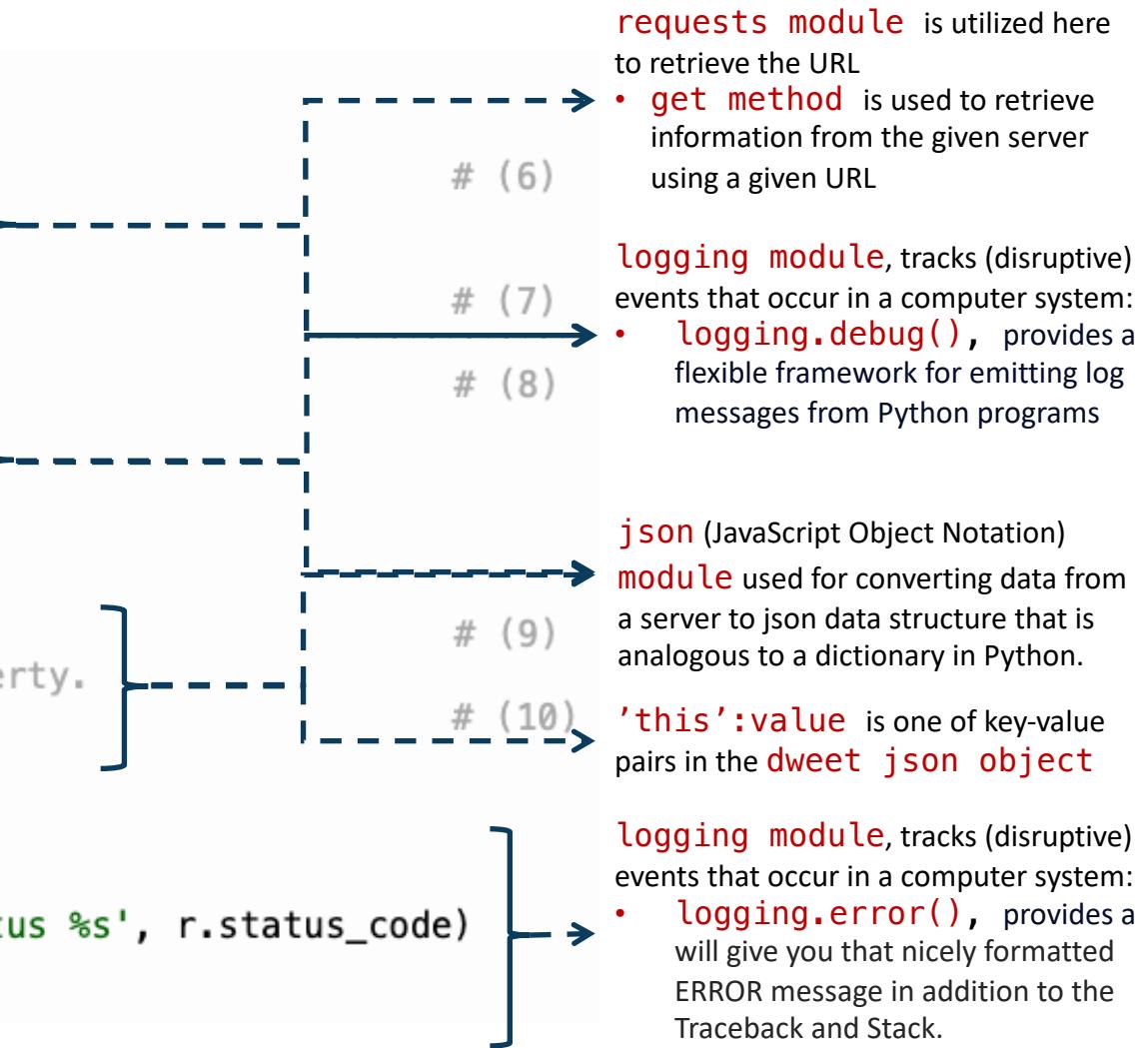
        dweet_content = None

        if dweet['this'] == 'succeeded':
            # We're just interested in the dweet content property.
            dweet_content = dweet['with'][0]['content']

    return dweet_content

else:
    logger.error('Getting last dweet failed with http status %s', r.status_code)
    return {}

```



```

def process_dweet(dweet):
    """Inspect the dweet and set LED state accordingly"""
    global last_led_state

    if not 'state' in dweet:
        return

    led_state = dweet['state']

    if led_state == last_led_state:
        return # LED is already in requested state.

    if led_state == 'on':
        led.on()
    elif led_state == 'blink':
        led.blink()
    else: # Off, including any unhandled state.
        led_state = 'off'
        led.off()

    if led_state != last_led_state:
        last_led_state = led_state
        logger.info('LED ' + led_state)

```

json (JavaScript Object Notation) module used for converting data from a server to json data structure that is analogous to a dictionary in Python.

- 'state':value is one of key-value pairs in the dweet json object

dweet['state'] can be initialized to 'on', 'off' or 'blink'

- 'state':value is one of key-value pairs in the dweet json object
- gpiozero module is used to initialize the state-value.

logging module, tracks (disruptive) events that occur in a computer system:

- logging.info() records if the state of the LED works as intended.



```

def poll_dweets_forever(delay_secs=2):
    """Poll dweet.io for dweets about our thing."""
    while True:
        dweet = get_latest_dweet()
        if dweet is not None:
            process_dweet(dweet)

        sleep(delay_secs)
    
```

When a **dweet** is available, from the `get_latest_dweet()` it is handled by `process_dweet()`.

# (11)

# (12)

# (13)

We `sleep` for a default of 2 seconds before continuing with the while-loop

```

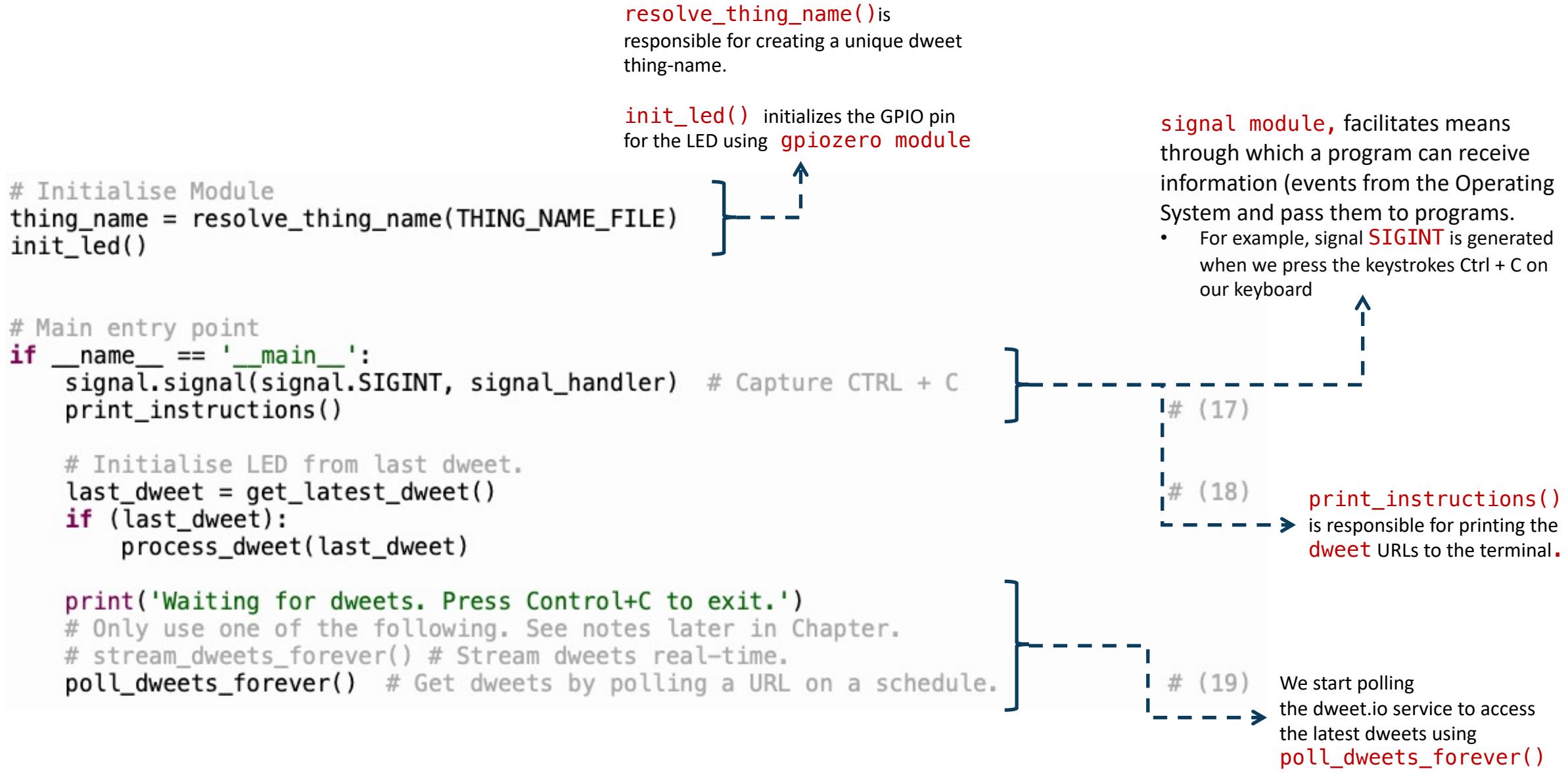
def print_instructions():
    """Print instructions to terminal."""
    print("LED Control URLs – Try them in your web browser:")
    print("  On   : " + URL + "/dweet/for/" + thing_name + "?state=on")
    print("  Off  : " + URL + "/dweet/for/" + thing_name + "?state=off")
    print("  Blink : " + URL + "/dweet/for/" + thing_name + "?state=blink\n")
    
```

`print_instructions()` is responsible for printing the `dweet` URLs to the terminal.

```

def signal_handler(sig, frame):
    """Release resources and clean up as needed."""
    print('You pressed Control+C')
    led.off()
    sys.exit(0)
    
```

`signal_handler()`, is responsible for turning off the LED.  
`sys.exit(0)`, with argument-0 raises an exception i.e., “successful termination”.



## Step-8:

- You should have completed your test runs
- You are still in **python-venv**
- **deactivate** your venv
- See example screenshot



A screenshot of a terminal window titled "pi@raspberrypi: ~/Desktop/Spring2023/LED\_examples". The window shows the command "deactivate" being run multiple times. The terminal interface includes a menu bar with File, Edit, Tabs, and Help, and a status bar at the bottom.

```
pi@raspberrypi:~/Desktop/Spring2023/LED_examples$ deactivate
(pienv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples$
```

```
(venv) pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $ deactivate
pi@raspberrypi:~/Desktop/Spring2024/dweet_LED_Example $
```

Congratulations!  
You just completed running your first IoT program!

Midterm projects  
- open discussions on how to proceed