MAE 6291
Internet of Things for Engineers

Prof. Kartik Bulusu, MAE Dept.

Week 5 [02/19/2024]	• Recap: Layers in IoT systems - 3 level layer model • Automating an email feature – Minimalist application layer in your IoT product design	• In-class email automation using yagmail and smtplib • Run dweet examples • In-class Flask API development • Discussion on what to expect in the remaining portion of the course

```
git clone https://github.com/gwu-mae6291-iot/spring2025_codes.git
```

GW School of Engineering & Applied Science
THE GEORGE WASHINGTON UNIVERSITY

Spring 2025

Photo: Kartik Bulusu

Midterm projects

Midterm Project Status – Spring 2024

Name	Project title	Hardware requirements	Status
Aleks Haskett	*Home Security System*	Pi Camera, Tracking sensor, Passive buzzer	**Approved. Needs to collect sensors**
Gerald Fattah	*Smart Animal Capture System*	IR Sensor, Pyroelectric ("Passive") InfraRed (PIR) module, Pi Camera	**Approved. Needs to collect sensors**
Jonathan Pang	*Proximity Alarm Door System (PADS)*	Pyroelectric ("Passive") InfraRed (PIR) module (HC-SR501), Bluetooth Tranceiver Module	**Approved. Needs to collect sensors**
Oliver Kristeya	*D ungeons & Dragans (D&D) Tower Roller*	Pi Camera, 3.5 in touch screen	**Need more information on 02/23, Approved on 02/26, Needs to collect sensors**
Talia Novack	*Dish Washer Helper*	Touch switch, analog heat sensor	**Approved. Need make and model numbers of the sensors**
Warren Nguyen	*Adaptive Lamp*	SenseHat, Photoresistor, Dimmable light sources	**Approved. Need make and model numbers of the sensors**
Selman Eris	*Food Scanner*	Pi Camera	**Approved. Needs to collect sensors**
Matthew Gouvin	*Plant Lighting Measurement Device*	Light sensors	**Approved. Need make and model numbers of the sensors**
Liza Mozolyuk	*Flight Tracking Interface*	SenseHat	**Need more information on 02/23, Approved on 02/26, Needs to collect sensors**
Bridget Orr	*Ukelele Tuner*	Sound sensor	**Approved. Need make and model numbers of the sensors**
Georgiana Mois	*MediTrack: Smark Medication Management*	Tilt Switch, Vibration Swtich	**Approved. Need make and model numbers of the sensors**
Alicia Ha	*Home Security Camera and Doorbell System*	Ultrasonic sensor, Pi Camera, PIR motion sensor, RGB LED, Passive Buzzer, Button	**Approved. Need make and model numbers of the sensors**
William Mai	*Cat Detector*	Pi NOIR Camera	**Approved. Needs to collect sensors**
Peter Wright	*Smart Cat Feeder*	Weight and Optical Sensor, actuator	**Approved. Need make and model numbers of the sensors**
Abdulrahman Alsaleh	*Camera by sensor detection*	Pi Camera, PIR motion sensor or Ultrasonic sensor,	**Approved. Need make and model numbers of the sensors**
Alvin Isaac	*Water Detection System*	Temperature, humidity and water level sensor	**Need more information on 02/23, Approved on 02/26, Needs to collect sensors**
Kartik Bulusu	*STREAM: **S**ensor s**T**ack fo**R** **E**nvironment**A**l **M**onitoring*	ESP 32, Barometer, GPS, SCD-30 - NDIR CO2 Temperature and Humidity Sensor	**Need more information; Unclear how he's going to pull this off!!**

Midterm Project Status – Spring 2025

Name	Project title	Hardware requirements	Status
Alexandra Trotter	Inconvenience in Animal Welfare	VIbration sensor – Recommended PIR sensor & Buzzer	**Approved. Needs to collect sensors**
Ben Sirota	Baja Car Speedometer	Hall effect Sensors, Photointerruptors - Recommndation: Pi Camera	**Approved. Needs to collect sensors Need more information**
Dominic Savarino	*EyePi: Intelligent Object Detection with Email Alerts*	Pi Camera	**Approved. Needs to collect sensors**
Elliot Hunter	*Smart Seat Occupancy and Safety System (Grad Student Perfector)*	Vibration Switch, Pi Camera, Relay, MQ2 Gas sensor, LED	**Approved. Needs to collect sensors**
Alex Vasilev	*Real-time battery display*	Voltage detector, A/D converter	**Approved. Needs to collect sensors**
Puchen Wang	*Infrared Sensor-Based Automatic Pet Door*	LEDs, IR transceivers – Recommendation: PIR sensors	**Approved. Needs to collect sensors**
Miya Liu	*iSwipe: connecting hungry students to meal swipes*	GPS module	**Approved. Needs to collect sensors Need more information**
Nick Neirotti	*Shade Runner 2025*	DC Motors, Motor Drivers, ESP32, Battery holder	**Approved. Needs to collect sensors**
Nathan Janssen	*Project Saver*	Recommendation: Servo motors	**Approved. Needs to collect actuators**
Omar Nayfeh	*REEFLEX Water Leak Detection System*	Sound sensor	**Conditionally Approved. Need make and model numbers of the sensors**
Shota Kakiuichi	*Smart Refill Monitoring System for Waste Management*	Ultrasonic sensor, LCD module, Active Buzzer, LED, button	**Approved. Needs to collect sensors**
William Lynam	PlantPal	Hiumiture sensors	**Approved. Need make and model numbers of the sensors**
Yazan Sawalhi	*Candle Monitor and Extinguisher*	Flame sensor, Ultrasound sensor., Motorized Fan	**Approved. Needs to collect sensors**
Aly Nguyen	*Extreme Study Buddy*	Servo Motors, LEDs, Sound sesnors, ultrasound sensor, LCD Module	**Approved. Need make and model numbers of the sensors**
Sumner Gubisch	*Print Schedular, Live Updates, Time-lapse and Controls for Vat Photopolymerization (VPP) Printers*	Pi Camera	**Approved. Need make and model numbers of the sensors**
Kartik Bulusu	*Translator-at-ease*	ESP 32, Barometer, GPS, SCD-30 - NDIR CO2 Temperature and Humidity Sensor	**Need more information; Unclear how he's going to pull this off!!**

Resetting the course for the next 5 weeks:

- Topics to be covered
- Weekly deliverables

Hardware:

1. ESP32
2. Cameras
3. SenseHat

Mathematics:

1. Basics of matrices
2. Applications of matrices: filters
3. Basics of Signal processing

App-development:

1. Flask
2. Micropython
3. Flask_Restful
4. WebSockets

Edge computing on the Pi:

Mathematics + Python + Signal processing

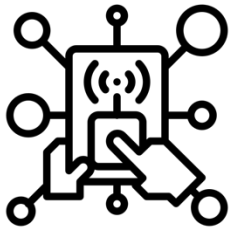
Expectations on student deliverables:

1. Midterm project demo
2. Midterm project presentation
3. Midterm project report in a conference-style template
4. Weekly coding HW
5. Weekly Quizzes
6. Final project proposal
7. Final project presentation
8. Final project demo
9. Final report in a conference-style template

Building up the IoT Architecture and Ecosystem

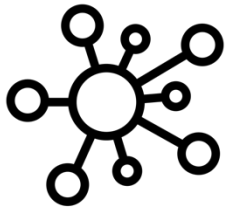
Icon Sources:
sensor by Carolina Cani:, sensor by Pham Duy Phuong Hung, sensor by Tippawan Sookruay, sensor by Lorenzo: https://thenounproject.com/browse/icons/term/sensor
fire sensor by LAFS : https://thenounproject.com/browse/icons/term/fire-sensor/
Ultrasound by Shocho: https://thenounproject.com/browse/icons/term/ultrasound/
Network by Solikin:, Network by Tippawan: https://thenounproject.com/browse/icons/term/network
application by Chaowalit Koetchuea: https://thenounproject.com/browse/icons/term/application/

Information-layer



Data

Communication-layer



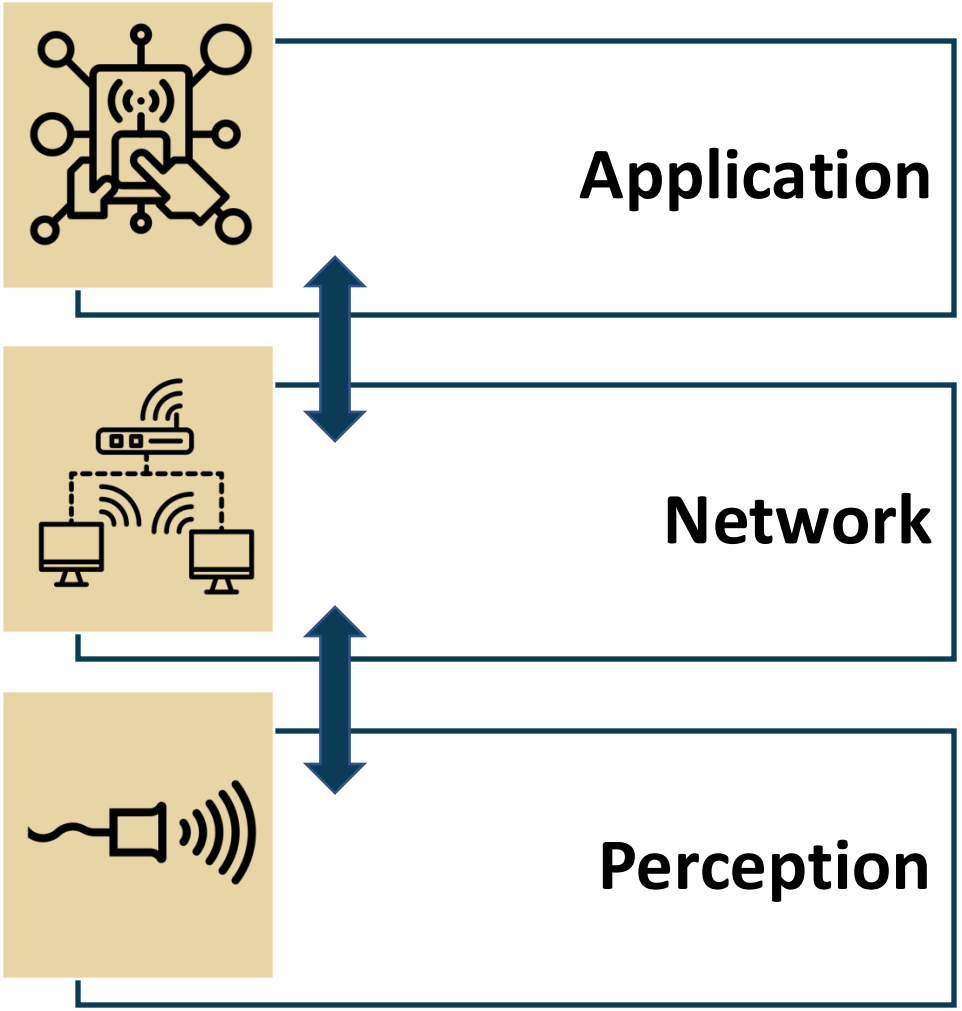
Connectivity

Sensor-layer

Things

The 3-Layer IoT Architecture

Icon Sources:
sensor by Carolina Cani:, sensor by Pham Duy Phuong Hung, sensor by Tippawan Sookruay, sensor by Lorenzo: https://thenounproject.com/browse/icons/term/sensor
wifi network by Matthias Hartmann:: https://thenounproject.com/browse/icons/term/wifi-network/
application by Chaowalit Koetchuea: https://thenounproject.com/browse/icons/term/application/



Automating an email feature – Minimalist application layer in your IoT product design

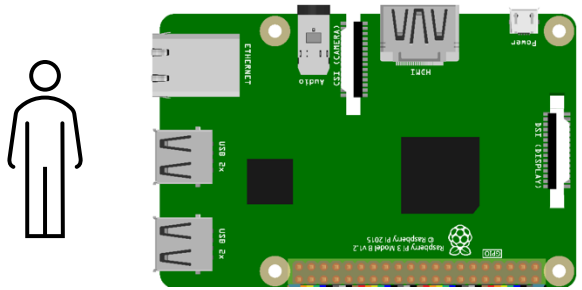
Graded Lab Activity (10 points)

Run
dweet_LED_Example

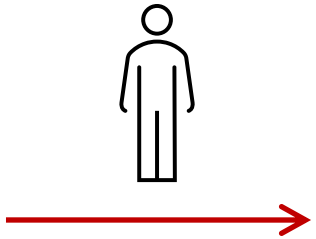
Your first IoT program

Graded Lab Activity (10 points)

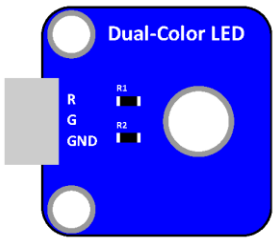
Scenario



I write some code on a computer and run it

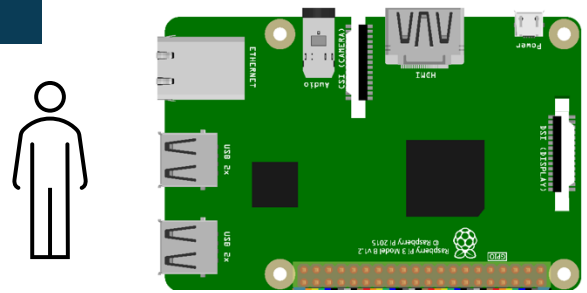


Communicate with a physical entity to do something

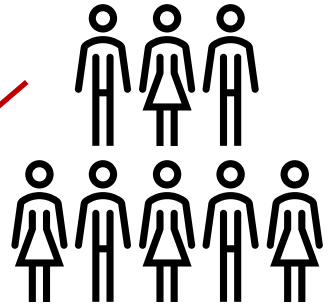


Observe response from that entity

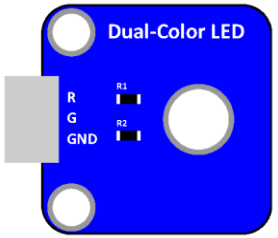
What-if-scenario



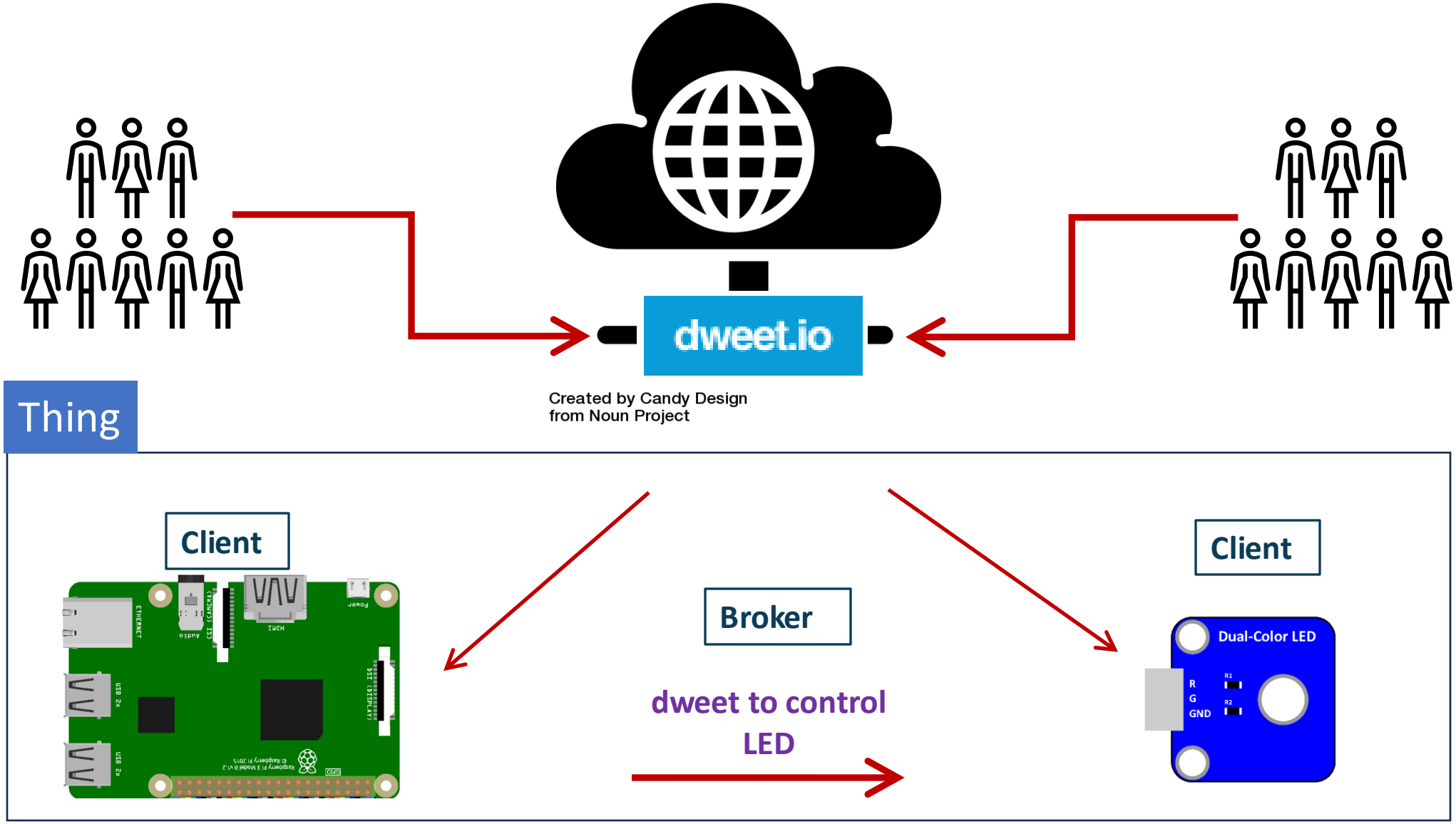
I write some code on a computer



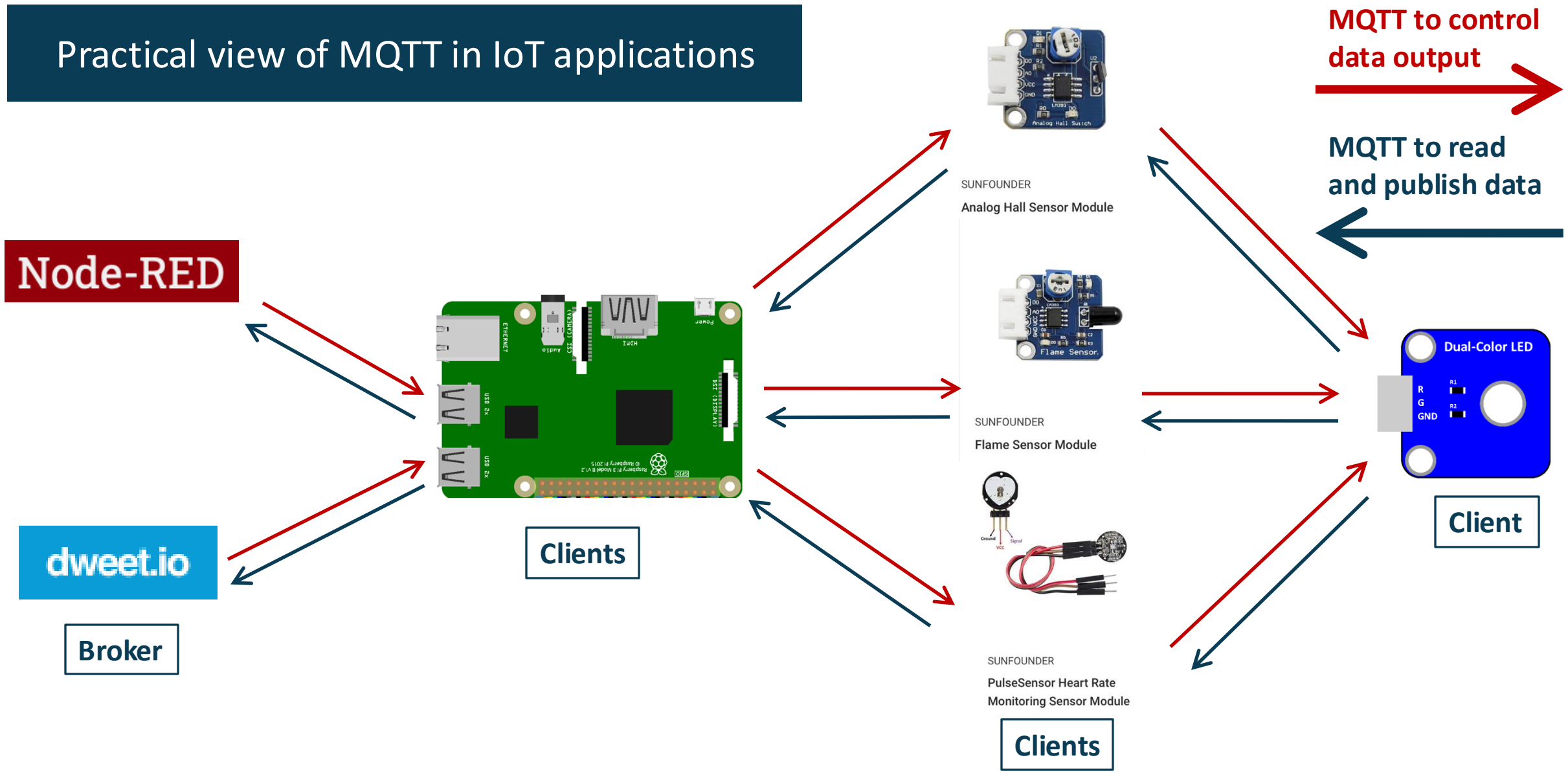
I realize people may not really care about it.
But want to use it anyway.



And curiously observe the response from that entity


dweet.io
Created by Candy Design
from Noun Project
Thing
Client
Broker
dweet to control
LED
Client
Dual-Color LED
R
G
GND

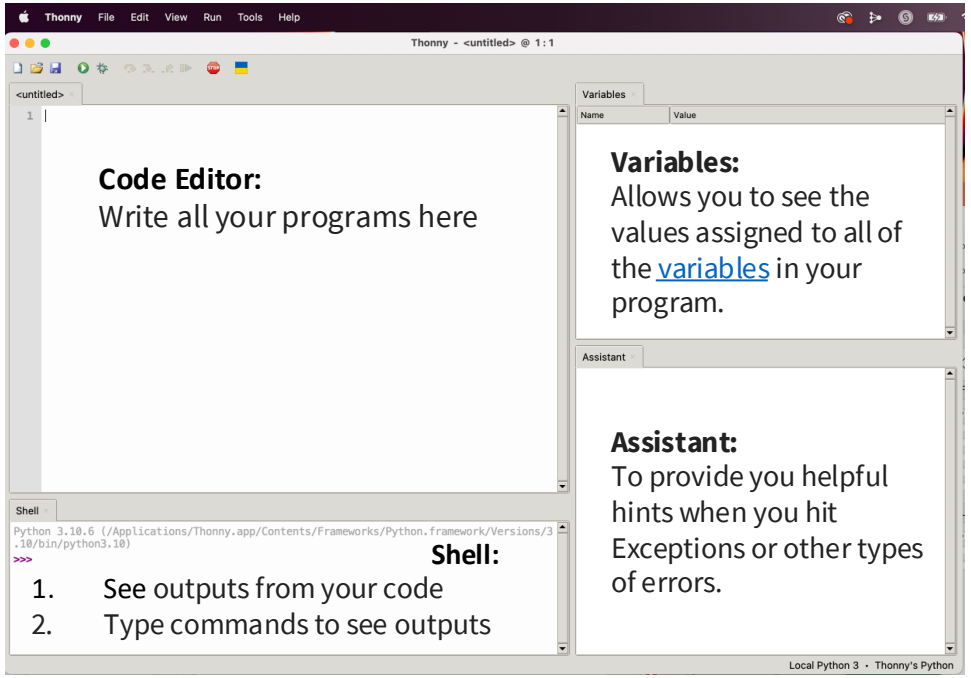
Practical view of MQTT in IoT applications



Goal-1:
Fetch dweet-data using Python

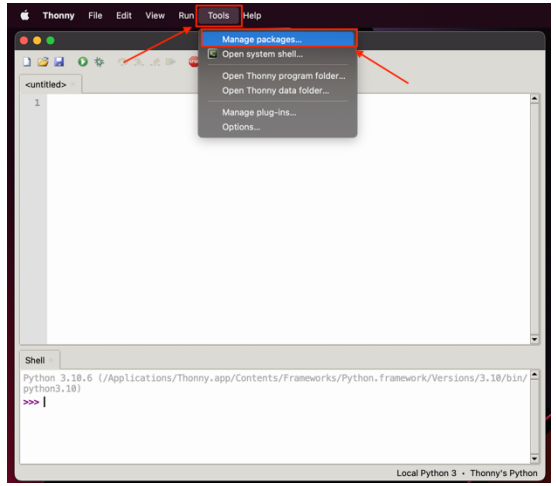
Step-0:
Install Thonny IDE

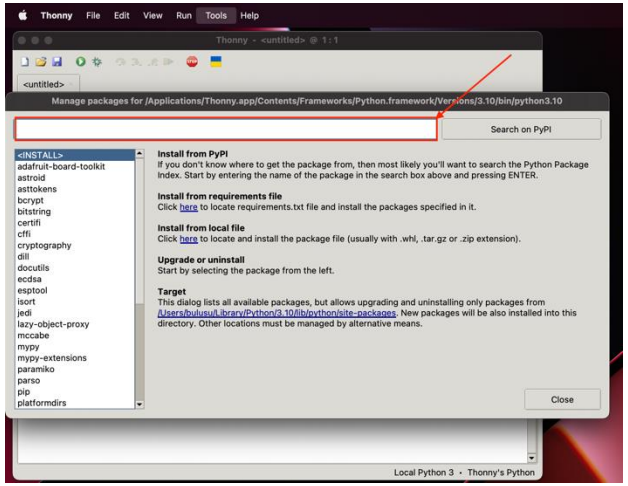
https://thonny.org

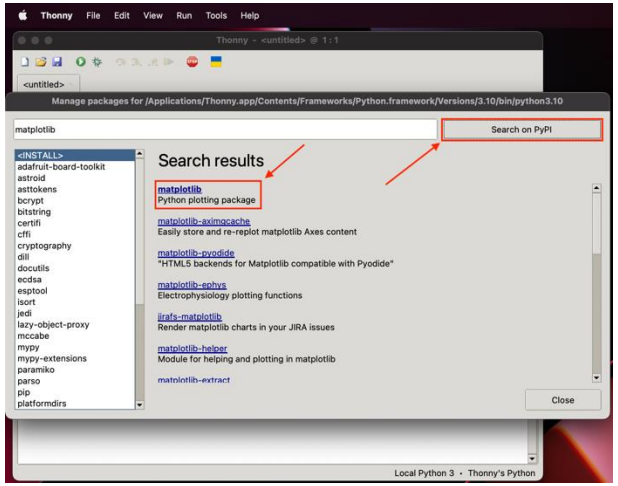


Step-1:
Install the following libraries:

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



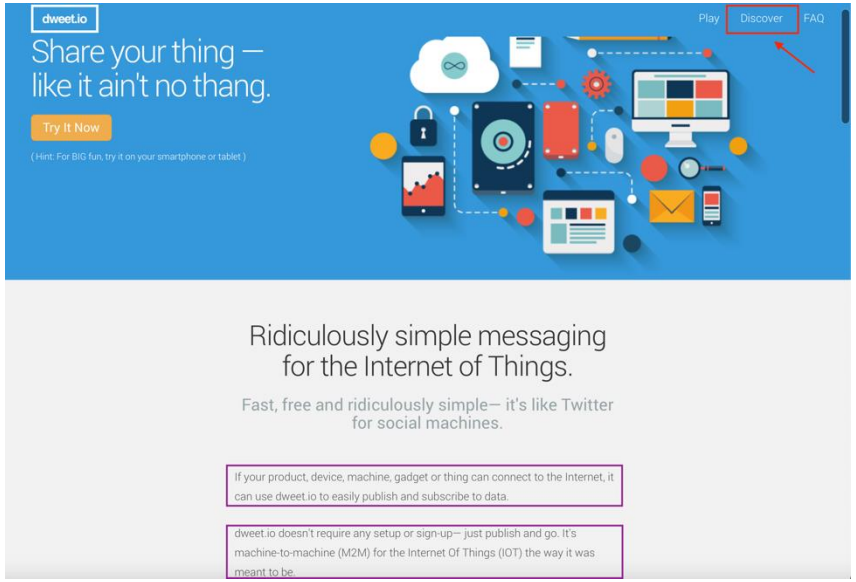


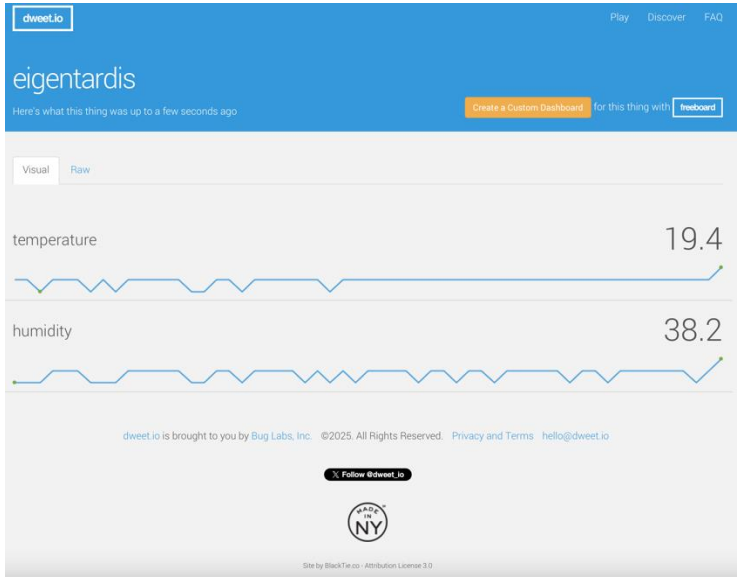


Step-2:
Go to dweet.io

dweet.io

https://dweet.io



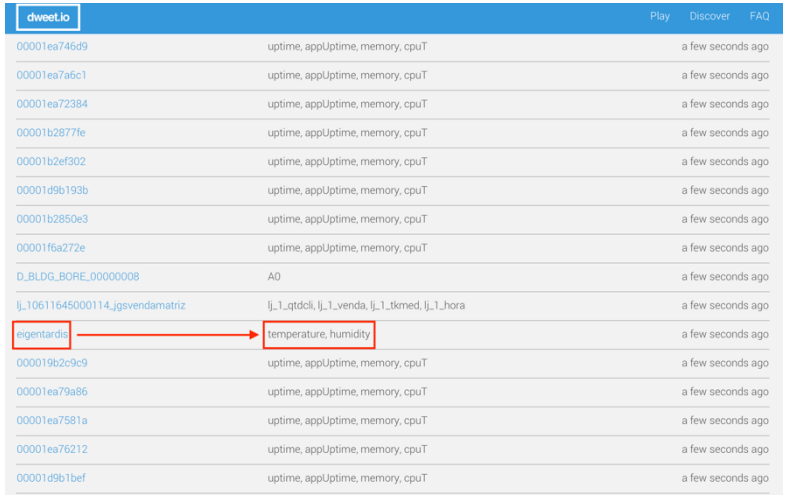


Ridiculously simple messaging for the Internet of Things.

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.



Step-3:
Let's write a Python program to retrieve dweet-data from a thing

Test drive:
https://dweet.io/follow/eigentardis

dweet-thing: eigentardis
data1: temperature
data2: humidity

dweetRetrieve.py

```python
# ================================================== #
# ============== Import libraries that matter ======== #
import dweepy
# ================================================== #

# ================================================== #

# Get thing name sourced from dweet.io
thing = input("Enter the thing name: " )

# Assign data variables
# Use the names copied verbatim from the thing
data1 = input("Retrieve IoT data #1 from the thing: " )

data2 = input("Retrieve IoT data #2 from the thing: " )

data3 = input("Retrieve IoT data #3 from the thing: " )

thing_info = dweepy.get_latest_dweet_for(thing)
# thing_info is a list of dictionaies

# ================================================== #


# ================================================== #
# ============== Start retrieving data ============== #

#Store "thing_info" as a dictionary; slice 0th entry
```

Variables

Name	Value
data1	'Temperature'
data2	'Humidity'
data3	'Barometric_Pressure'
dateNow	'2023-02-24'
date_created	'2023-02-24T02:04:30.933Z'
dweepy	<module 'dweepy' from '/Users/bulusu/Li
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-
this_dict	{'thing': '4C_Seattle1', 'created': '2023-0

Assistant

The code in dweetRetrieve.py looks good.

If it is not working as it should, then consider using some general debugging techniques.

Was it helpful or confusing?

Shell

```
>>> %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
```

Local Python 3 • Thonny's Python

Step-1:

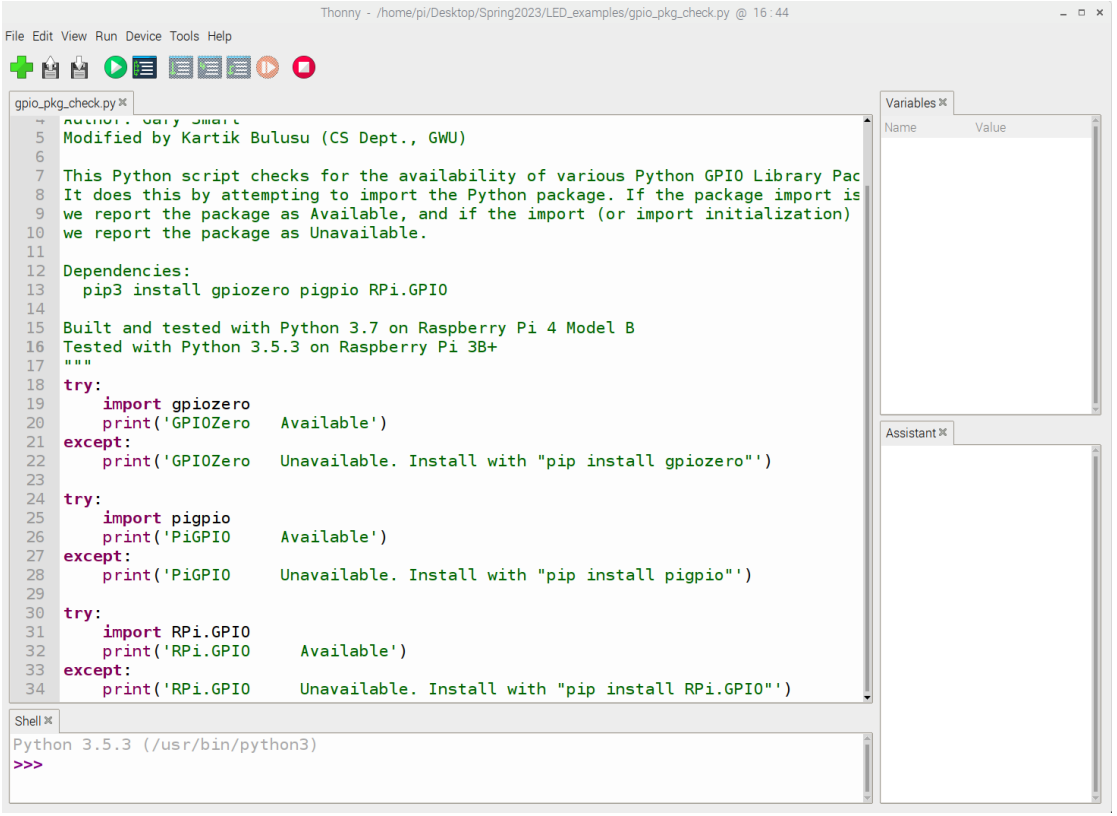
- Create python virtual environment
  - Activate python-venv
  - Install pip

```
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



(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

```
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 f
or 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 $
```

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

Step-4:

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

- **matplotlib**
- **numpy**
- **yagmail**

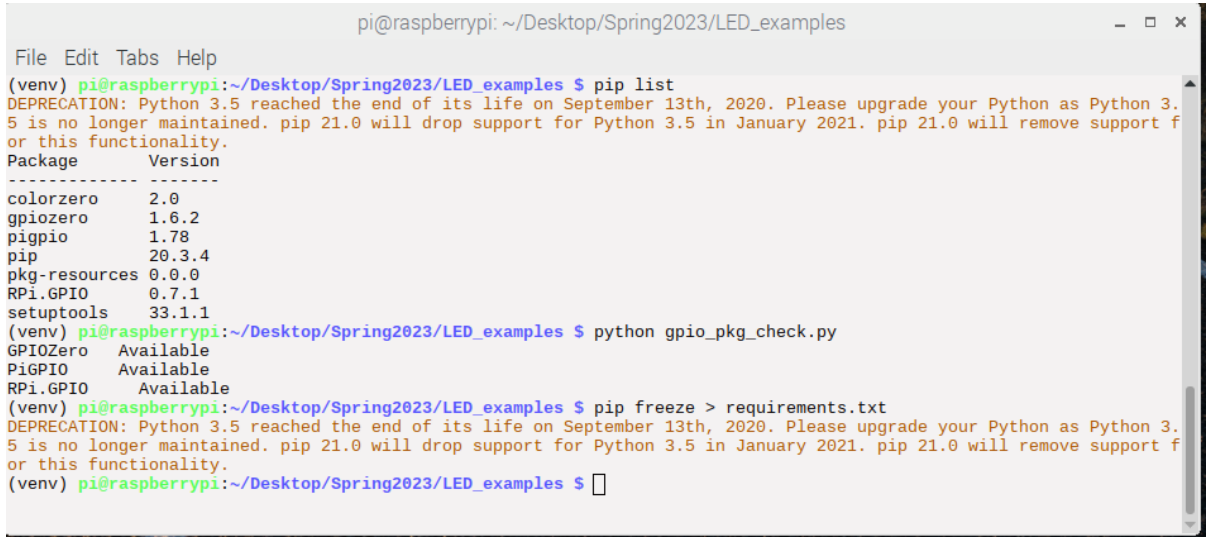
Note:

- **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



(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
```

Import libraries that are relevant for interaction with the Raspberry Pi hardware such as GPIO pins, camera ports etc.

```
GPIO.setmode(GPIO.BOARD)

GPIO.setup(12, GPIO.OUT)
```

Set up GPIO pins as data outputs or inputs for sensors and actuators

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

Create user-defined functions to modularize your code and make it easy to work.

Create user-defined functions to release the GPIO pins

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

Create entry point into the program and pull all functions in

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

```
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
```

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

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

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.

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/

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

```
# 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


# 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)


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

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

gpiozero module is a simple interface to GPIO devices with Raspberry Pi, developed and maintained by Ben Nuttalland Dave Jones.

- It is a simpler alternative to RPi.GPIO module
- Documentation: https://gpiozero.readthedocs.io/en/latest/

```python
# 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
```

gpiozero 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)

(5)

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

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

```python
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 {}
```

requests module is utilized here to retrieve the URL

- get method is used to retrieve information from the given server using a given URL

(6)

(7)

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

- logging.debug(), provides a flexible framework for emitting log messages from Python programs

(8)

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

(9)

(10)

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

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

- logging.error(), provides a will give you that nicely formatted ERROR message in addition to the Traceback and Stack.

```python
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

(14)

(15)

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.

(16)

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

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

```python
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

```python
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.

```python
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".

resolve_thing_name()is responsible for creating a unique dweet thing-name.

init_led() initializes the GPIO pin for the LED using gpiozero module

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

```
# Initialise Module
thing_name = resolve_thing_name(THING_NAME_FILE)
init_led()


# Main entry point
if __name__ == '__main__':
    signal.signal(signal.SIGINT, signal_handler)  # Capture CTRL + C
    print_instructions()

    # Initialise LED from last dweet.
    last_dweet = get_latest_dweet()
    if (last_dweet):
        process_dweet(last_dweet)

    print('Waiting for dweets. Press Control+C to exit.')
    # Only use one of the following. See notes later in Chapter.
    # stream_dweets_forever() # Stream dweets real-time.
    poll_dweets_forever()  # Get dweets by polling a URL on a schedule.
```

(17)

(18)

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

(19)

We start polling the dweet.io service to access the latest dweets using poll_dweets_forever()

Step-8:

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

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

pi@raspberrypi:~$/Desktop/Spring2024/dweet_LED_Example $

Congratulations!
You just completed running you first IoT program!

Midterm projects
- open discussions on how to proceed

Source:
https://flask.palletsprojects.com/en/2.2.x/
https://en.wikipedia.org/wiki/Plotly
javascript file by SAM Designs from https://thenounproject.com/browse/icons/term/javascript-file/
html by tezar tantular from https://thenounproject.com/browse/icons/term/html/

Demo project:
Create a Flask API for IoT Applications
[Graded Lab Activity]

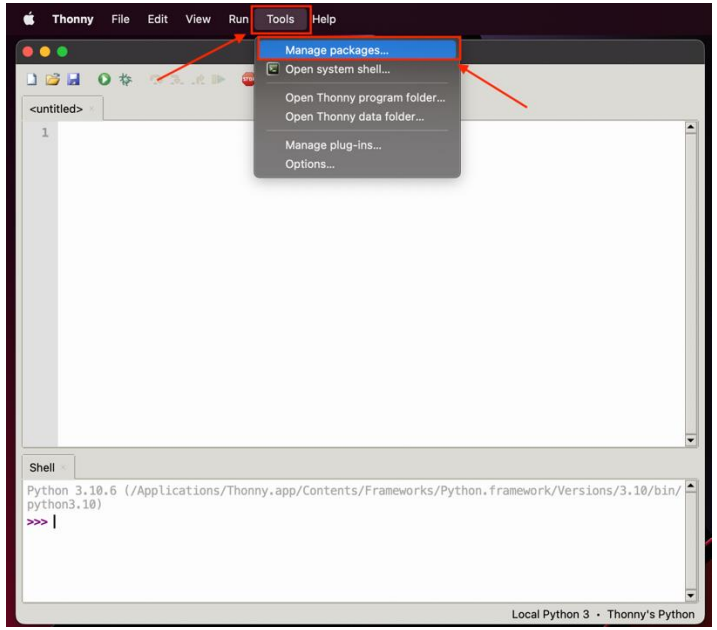
1. Python 3 with any IDE or terminal
2. Familiarity with flask API
3. Basic HTML with JavaScript
4. Familiarity with Plotly

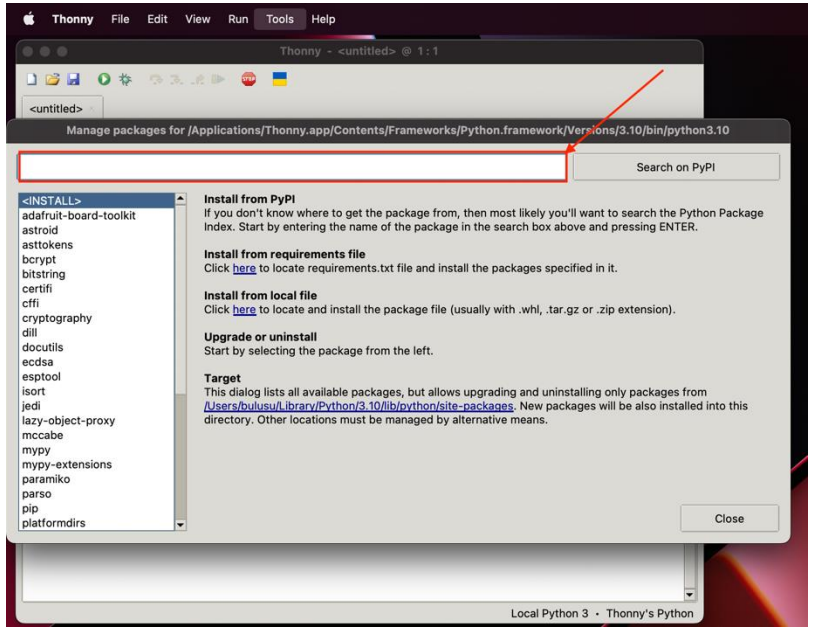
Source:
https://flask.palletsprojects.com/en/2.2.x/
https://en.wikipedia.org/wiki/Plotly

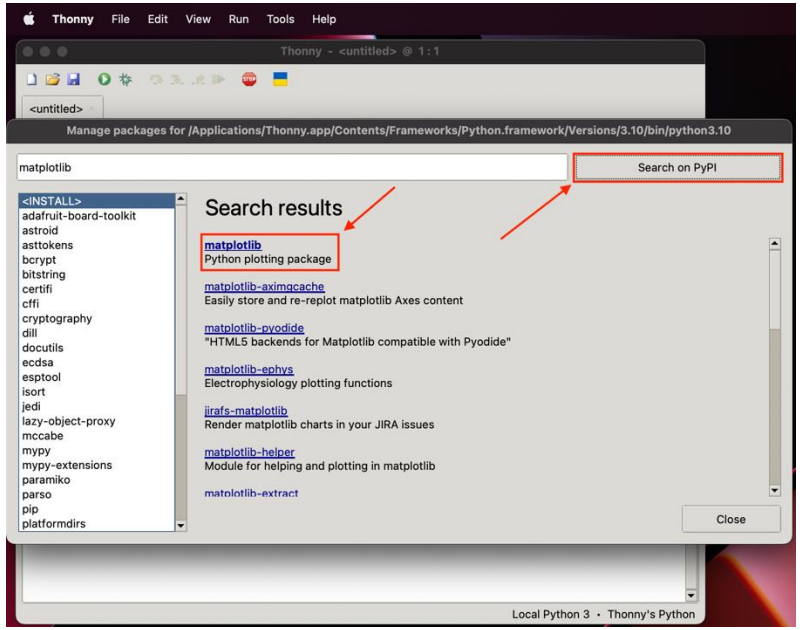
- **Flask** is a micro web framework written in Python.
- **Components:**
  - ***Werkzeug:*** Utility library for Web Server Gateway Interface (WSGI) applications
  - ***Jinja:*** Template engine similar to Django that handles templates in a sandbox
  - ***MarkupSafe:*** String handling library
  - ***ItsDangerous:*** Safe data serialization library

- **Plotly** provides online graphing, analytics, and statistics tools
  - for individuals and collaboration,
  - as well as scientific graphing libraries for Python, R, MATLAB, Perl, Julia, Arduino, and REST.
- **Open-source products:**
  - ***Dash:*** Open-source framework for building web-based analytic applications.
  - ***Chart Studio Cloud:*** Free, online tool for interactive graphics
  - ***Plotly.js data visualization JavaScript library*** for creating graphs and powers Plotly.py for Python, as well as Plotly.R for R, MATLAB, Node.js, Julia, and Arduino and a REST API

Installing packages in Thonny







Check or install the following libraries in Python 3.10.11: (Note these are the bare minimum versions)

Flask 2.3.1
plotly 5.15.0
simplejson 3.19.1

pandas 2.0.2
numpy 1.24.2
matplotlib 3.7.1

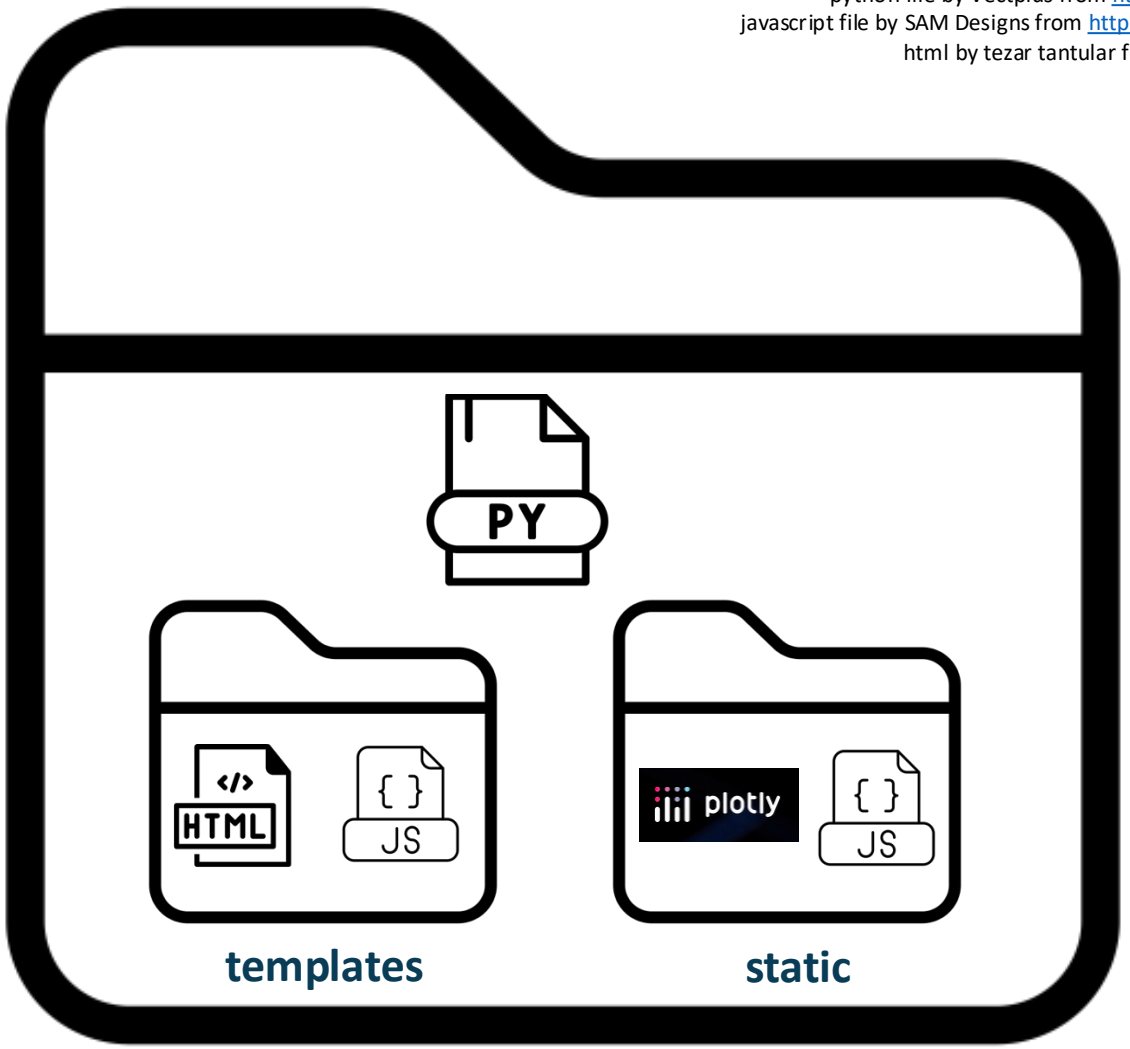
datetime 5.0
time 1.0.0

Alternative:

```
>>> pip install <package_name>
>>> # Or install using pip3
>>> # in your virtual environment
```

Source:
Folder by Colourcreatype from https://thenounproject.com/browse/icons/term/folder/
python file by Vectplus from https://thenounproject.com/browse/icons/term/python-file/
javascript file by SAM Designs from https://thenounproject.com/browse/icons/term/javascript-file/
html by tezar tantular from https://thenounproject.com/browse/icons/term/html/
https://en.wikipedia.org/wiki/Plotly

Essentials: Files and folders needed to create the intended APP



Skeleton of the Python program for a flask-server on the Raspberry Pi

Source:
https://flask.palletsprojects.com/en/2.2.x/

Step-1:
Initialize variables and setup Flask instance

```
from flask import Flask, request, render_template
from flask_restful import Resource, Api, reqparse, inputs
import pandas as pd
import json
import plotly
import plotly.subplots
import plotly.express as px
import random
import numpy as np
import matplotlib.pyplot as plt
import time
import datetime
import logging
import thing_file
```

Import libraries that are relevant for interaction with the Raspberry Pi hardware such as

flask,
json,
plotly and its derivatives
pandas
numpy
matplotlib etc.

Source:
Folder by Colourcreatype from https://thenounproject.com/browse/icons/term/folder/
python file by Vectplus from https://thenounproject.com/browse/icons/term/python-file/
javascript file by SAM Designs from https://thenounproject.com/browse/icons/term/javascript-file/
html by tezar tantular from https://thenounproject.com/browse/icons/term/html/
https://en.wikipedia.org/wiki/Plotly

Step-2: Initialize variables and setup Flask instance

```
# ======== Initialize Logging ===========

# Global logging configuration
logging.basicConfig(level=logging.WARNING)

# Logger for this module
logger = logging.getLogger('main')

# Debugging for this file.
logger.setLevel(logging.INFO)
```

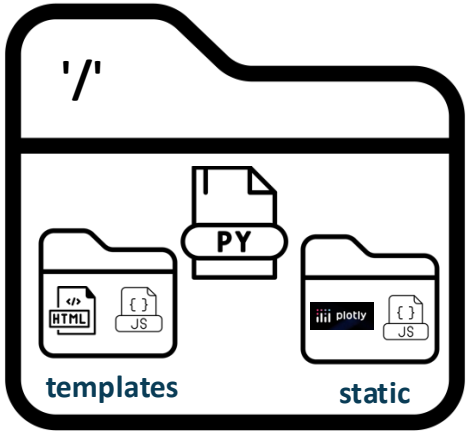
Provides warning on any components that work within flask or other imported libraries

Reference:
https://docs.python.org/3/howto/logging.html

thing_file.py
Logging libraries and a few more are place in this custom library provided to you

```
# ==== Flask & Flask-RESTful instance variables ====

# Core Flask app.
app = Flask(__name__)

# ==== Flask & Flask-Restful Related Functions ====
# @app.route applies to the core Flask instance (app).
# Here we are serving a simple web page.
@app.route('/' + thing_file.thing_name)
```

Initiate flask server-related variables and functions



Step-3:
Create a function notdash() to return JSON-formatted data to an html frontend

```
def notdash():
  global data
  data = {
    'timeT': [],
    'Voltage': []
  }
```

Dictionary of empty lists that get appended with data

```
    # Create the graph with subplots
      fig = plotly.tools.make_subplots(rows=1, cols=1, vertical_spacing=0.2)
      fig['layout']['margin'] = {
        'l': 30, 'r': 10, 'b': 30, 't': 10
      }
```

Dictionary of figure layout that is transferred to the html frontend with plotly, JavaScript embedded in it

```
    for i in range(20):
            data['Voltage'].append(random.randint(0, 100))
            data['timeT'].append(timeT)

        fig.append_trace({
          'x': data['timeT'],
          'y': data['Voltage'],
                'mode': 'lines+markers',
          'type': 'scatter'
        }, 1, 1)
```

Loop to generate random data and plot it in a trace that is transferred to the html frontend with plotly, JavaScript embedded in it

```
graphJSON = json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)
return render_template('notdash.html', graphJSON=graphJSON)
```

json.dumps will convert a subset of Python objects into a json
render_template tells Flask to use an HTML template

Step-5
Create a function notdash() to return JSON-formatted data to an html frontend

```
if __name__ == '__main__':

    # If you have debug=True and receive
    # the error "OSError: [Errno 8] Exec format error", then:
    # remove the execuition bit on this file from a Terminal, ie:
    # chmod -x flask_api_server.py
    #
    # Flask GitHub Issue:
    # https://github.com/pallets/flask/issues/3189

    app.run(host="0.0.0.0", debug=True)
```

Creates entry point into the program and executes app.run() in debug mode.

debug=False:
translates to developer mode

app.run() renders the webpage with data plots on a

local host:
127.0.0.1:5000

Port:5000 is a default

The host address can be changed to the IP address of the server.

Skeleton of the the basic HTML code to display data from your flask-app

Sources:
https://plotly.com
API by Vectors Point from https://thenounproject.com/browse/icons/term/api/
json by ME from https://thenounproject.com/browse/icons/term/json/
javascript file by SAM Designs from https://thenounproject.com/browse/icons/term/javascript-file/
html by tezar tantular from https://thenounproject.com/browse/icons/term/html/
https://towardsdatascience.com/web-visualization-with-plotly-and-flask-3660abf9c946

```
<!doctype html>
<html>

<head>
        <meta http-equiv="refresh" content="10">
</head>

<body>
        <h1>Prof. Kartik Bulusu's sensor data</h1>
        <div id='chart' class='chart'"></div>
</body>

<!-- <script src='https://cdn.plot.ly/plotly-latest.min.js'></script> -->

<script src='/static/plotly-latest.min.js'></script>

<script type='text/javascript'>
        var graphs = {{graphJSON | safe}};
        Plotly.plot('chart',graphs,{});
</script>

</html>
```

Will refresh the page every 10 seconds

Webpage title etc

Location of plotly-latest.min.js

To download: https://plotly.com/javascript/getting-started/

{{graphJSON | safe}}: Injects a variable that came from the server directly in the JavaScript code.

Plotly.plot(): Creates a **line chart** drawn into a element on the page, with data from **graphs** with **layout** provided by the server

HW on Ultrasound sensor due on 02/26/2025 (10 points)

git clone https://github.com/gwu-mae6291-iot/spring2025_codes.git

Goal-1

Set up a Python virtual environment
- “Sandbox” your Python project

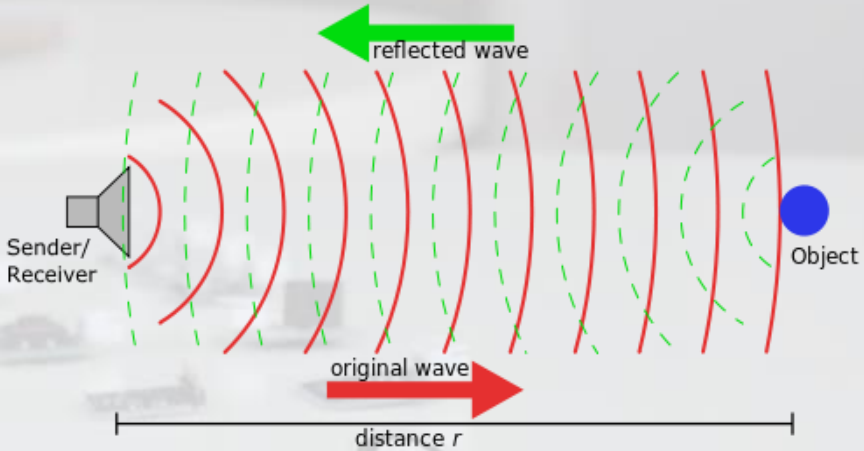
Goal-2

Demonstrate your python script at boot
Using cron, the UNIX scheduler

Ultrasound Signals and its Applications

Icon Sources:
sensor by Carolina Cani:, sensor by Pham Duy Phuong Hung, sensor by Tippawan Sookruay, sensor by Lorenzo: https://thenounproject.com/browse/icons/term/sensor
fire sensor by LAFS : https://thenounproject.com/browse/icons/term/fire-sensor/
Ultrasound by Shocho: https://thenounproject.com/browse/icons/term/ultrasound/
Network by Solikin:, Network by Tippawan: https://thenounproject.com/browse/icons/term/network
application by Chaowalit Koetchuea: https://thenounproject.com/browse/icons/term/application/

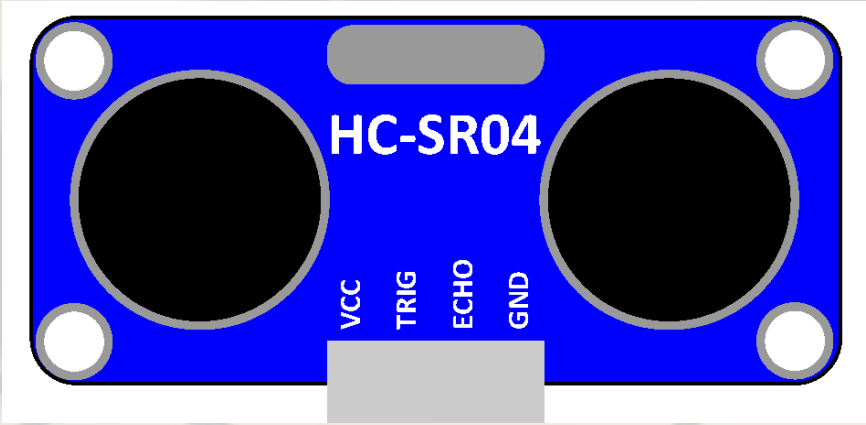
Source: https://youtu.be/J-VVh5ezqGA?si=G9iu055uV3MinyUA



$$Distance\ traversed = (Speed\ of\ sound) \times (Time\ elapsed/2)$$

Photo: Kartik Bulusu

Know your Ultrasonic Sensor



The Ultrasonic sensor sends out ultrasonic waves to detect objects and measure distances.

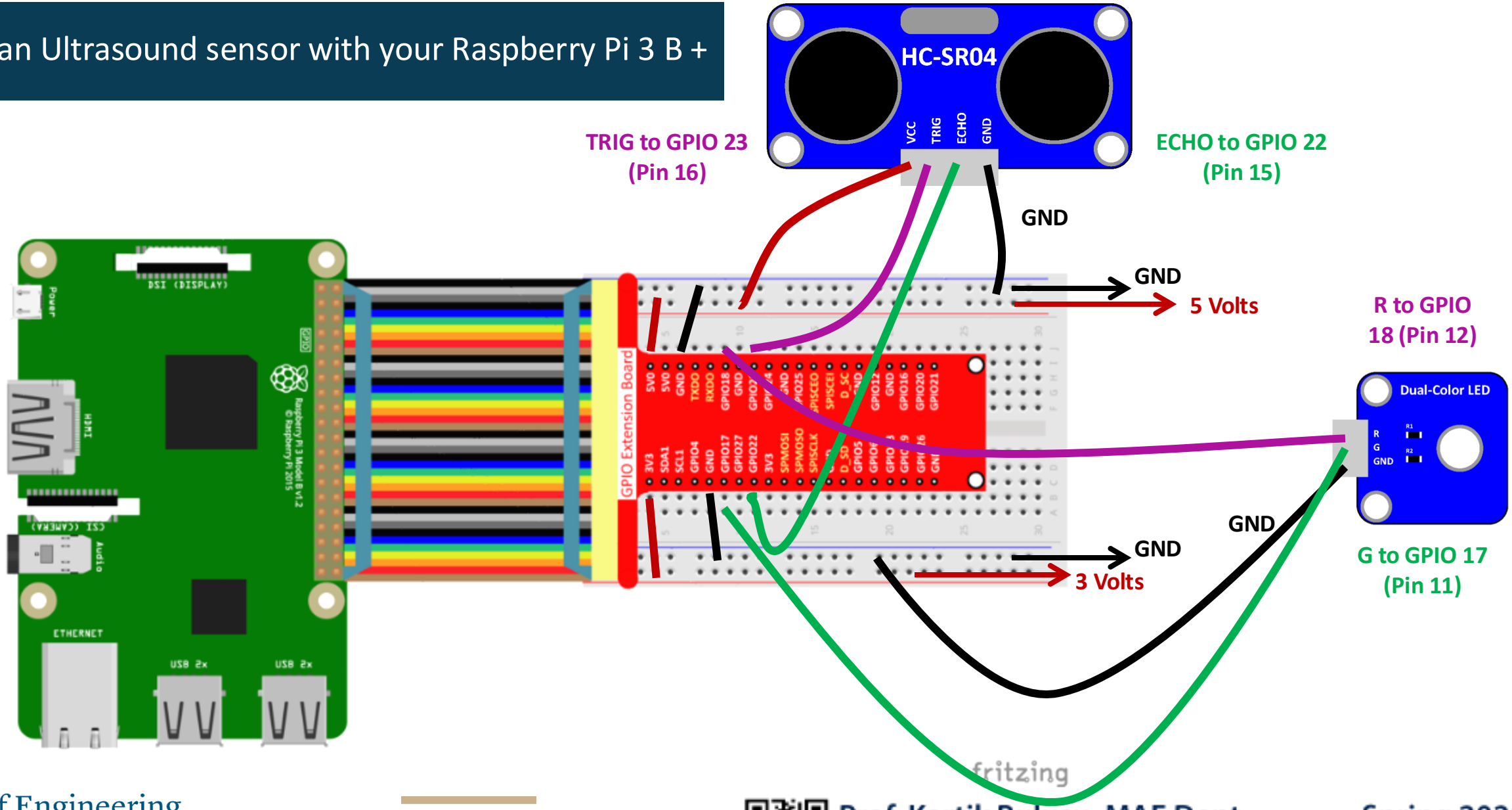
Connectors:
4-pin connector wires
Or in some cases 5-pin connector wires

Goal of this HW segment:

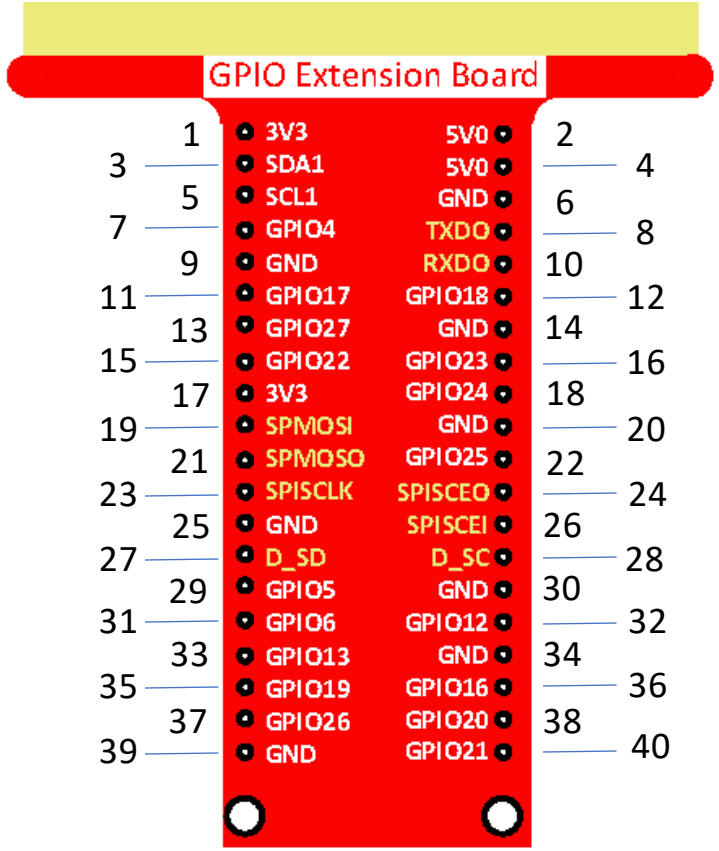
- Co-work (if you need to)
  - Observe, ask and try in groups
- Make
  - Build-a-hack
  - Ultrasound sensors and Raspberry Pi 4B boards
- Analyze data using Python

Photo: Kartik Bulusu

Start an Ultrasound sensor with your Raspberry Pi 3 B +



Pseudo-code to kick start your Raspberry Pi Model 3 B+ (RPi)




```
import LIBRARY as NAME
import ANOTHER_LIBRARY
```

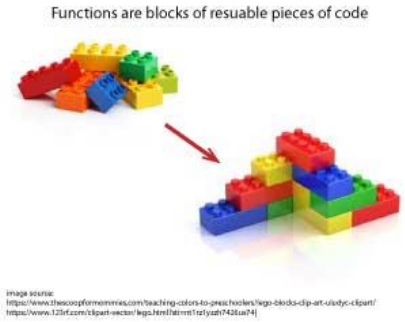
```
INITIALIZE GPIO CHANNELS

DEFINE SETUP FUNCTION
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(CHANNEL-1, GPIO.OUT)
    GPIO.setup(CHANNEL-2, GPIO.IN))
```

```
DEFINE DISTANCE FUNCTION
return (TIME_ELAPSED / 2) * 340 * 100
```

```
DEFINE LOOP FUNCTION
    while True:
        ...
```

```
DEFINE DESTROY FUNCTION
    CLEAN UP GPIO CHANNELS
```

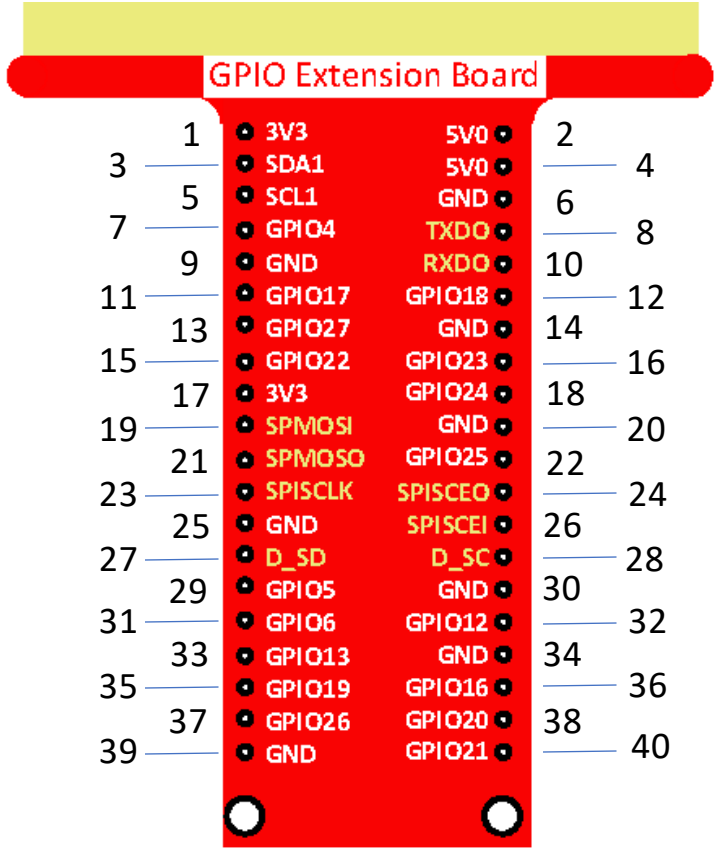
User defined functions




```
if __name__ == "__main__":
    setup():
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

Entry point into the program – pulls in all user defined functions

A simple python code to kick start your Raspberry Pi Model 3 B+ (RPi)




```python
import RPi.GPIO as GPIO
import time
```

```python
TRIG = 16
ECHO = 15
def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(TRIG, GPIO.OUT)
    GPIO.setup(ECHO, GPIO.IN))
```

```python
def distance():
    GPIO.output(TRIG, 0)
    time.sleep(0.000002)
    GPIO.output(TRIG, 1)
    time.sleep(0.00001)
    GPIO.output(TRIG, 0)

    while GPIO.input(ECHO) == 0:
        time1 = time.time()

    while GPIO.input(ECHO) == 1:
        time2 = time.time()

    during = time2 - time1
    return (during / 2) * 340 * 100
```

```python
def loop():
    while True:
        dist = distance()
        print(dist, 'cm')
        print('')
        time.sleep(0.1)
```

```python
def destroy():
    GPIO.cleanup()
```

```python
if __name__ == "__main__":
    setup():
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```