Expt-3 Data Acquisition and Processing.

Objective

To Study methods of Data handling for Data acquisition and Data processing and perform.
Data Acquisition and Processing for Weather Monitoring using Sensors and API.
Theory: .

3.1 Data handling

In general, the data handling process has three major stages: data collection (or acquisition), data processing, and presentation (output). However, there can also be substages like data organization, cleaning, and so on based on the type of data and scenarios where it is being handled. For example, data cleaning is widely used in most data science projects where the data is not structured like email or flat files.

Data acquisition

For data acquisition, first, we need to know about various data sources. We can categorize different data sources in the following five categories:

•**Field data**: It comes via various devices like sensors, IoT devices, PLC, and so on.

•**Local files**: Files stored locally like CSV, XML, JSON, and so on.

•**Databases**: Databases like MongoDB, SQLite, and so on. Store organization's business data.

•**APIs**: To get data from other platforms of services like AccuWeather API.

•**Public datasets**: Reference bulk data, available publicly to download. In the case of Edge/IoT, we majorly deal with field data acquisition. The sensors generate data values (for example, temperature) at a time interval. The combination of data value and its timestamp is referred to as *timeseries data* whi ch is structured data. Along with the data value and time, there can also be more information, like the serial number of the sensor, its location, and so on. In most industrial setup or IIoT scenarios, the sensor's data does not directly transmit to Edge or central computer; first, it is trans- mitted to PLC, and then Edge or central computer collects it from PLC. The figure 1 shows both the situation of direct connection of sensors and indirect via PLC to Edge device:

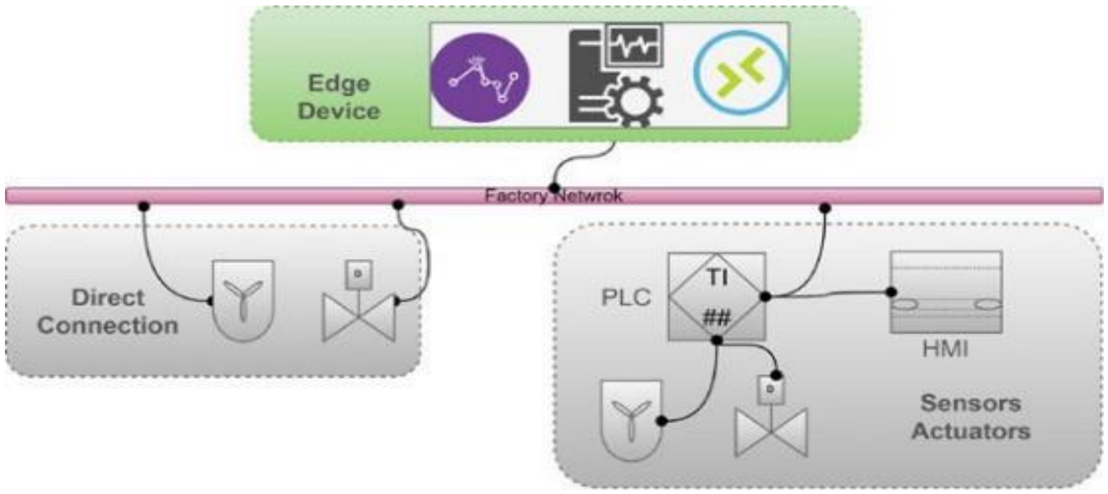


Figure 1: *Direct sensors data Vs. via PLC*

Data processing and output handling

Data processing generally means converting the acquired data into meaningful information or output via the application program. For example, after acquiring a patient's ECG data, the program should be able to identify critical or everyday situations.

Output handling After data processing, the output shall be presented to the user or another program/service in an understandable and consistent format. For example, after processing the patient's ECG data, the application program can raise an alert via flashing a red LED, raising an alarm sound over the speaker, and so on. The same ECG data can also be plotted over a trend view. The following figure shows an Edge application performing data handling tasks:

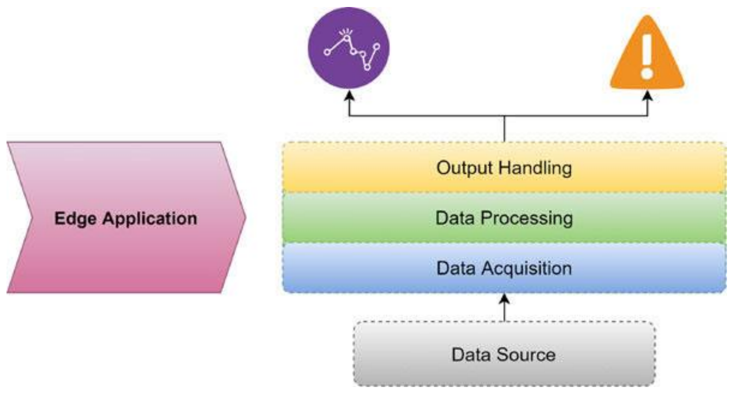


Figure : *Data Handling*

Python data handling : Python has a very rich set of libraries for data handling. We will cover data handling via external APIs.

<u>EXAMPLE-1: WEATHER MONITORING USING EXTERNAL API.</u>

This example presents acquiring data from an external API, processing it, and displaying meaningful information. We are using publicly available free weather APIs from OpenWeather [**<u>https://openweathermap.org/appid</u>**]. The external API is not part of our application or system but is hosted by some external system. The external system exposes an endpoint for the API. Any- one who knows the endpoint can send a request to execute this API and receive the response or result. Calling the exposed API over HTTP (some- times referred to as REST API) is common and covered in this example. For this, we are using Python's **request** module from the **urllib** library, which interacts with URLs. The following figure shows the interaction between a Python client program and a server (API Host):

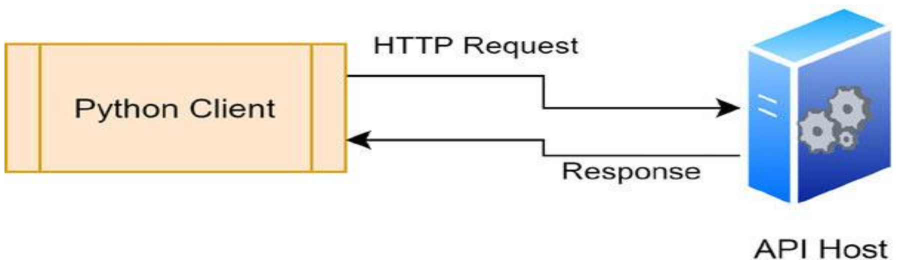
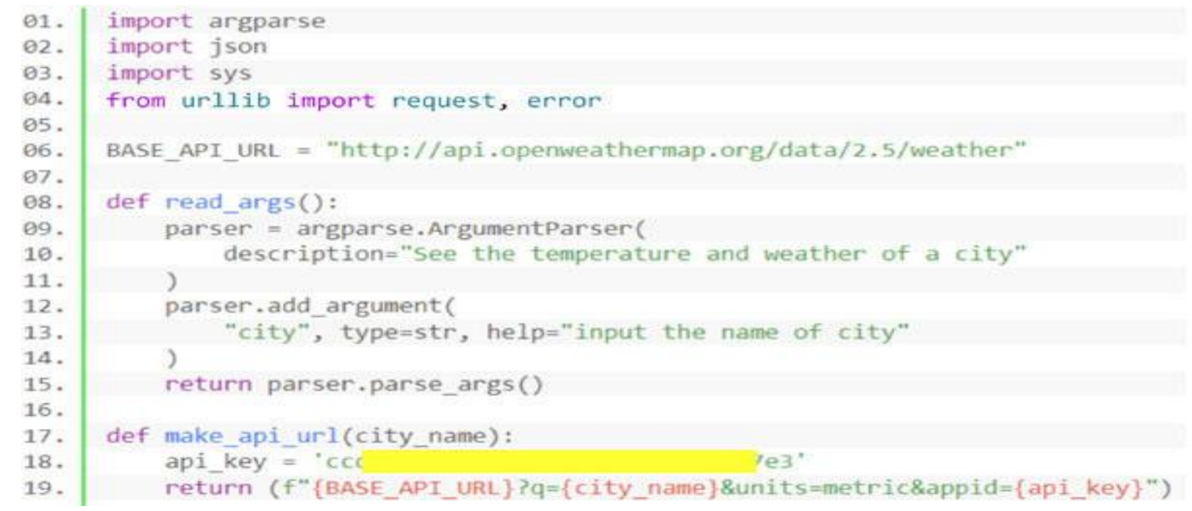


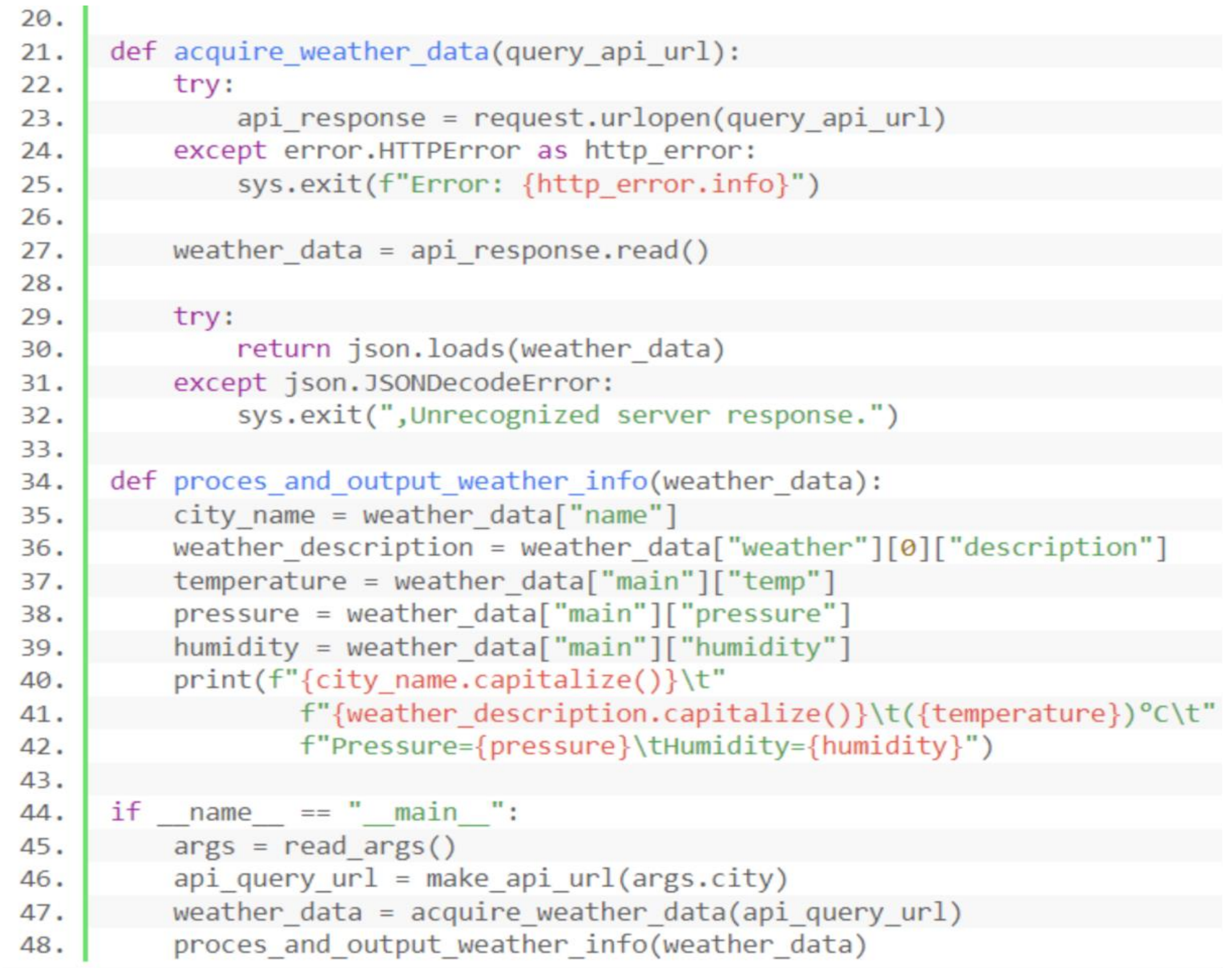
Figure : *Interaction between Python Client and API over HTTP*

First, you need to signup with OpenWeather on this link **https:// openweathermap.org/home/sign_up** and get the API key which will be sent to your email id on successful sign-up. In general, the API key is used to ensure that only authorized users can call the APIs; any arbitrary calls to APIs will not be served. After getting the API key, open VS Code set new folder, create Pythonvirtual environment, run **pip install urllib** to install **urllib**, create a new file, name it **weather-status.py**, and enter the following code:



```python
import argparse
import json
import sys
from urllib import request, error

BASE_API_URL = "http://api.openweathermap.org/data/2.5/weather"

def read_args():
    parser = argparse.ArgumentParser(
        description="See the temperature and weather of a city"
    )
    parser.add_argument(
        "city", type=str, help="input the name of city"
    )
    return parser.parse_args()

def make_api_url(city_name):
    api_key = 'cc[redacted]e3'
    return (f"{BASE_API_URL}?q={city_name}&units=metric&appid={api_key}")
```

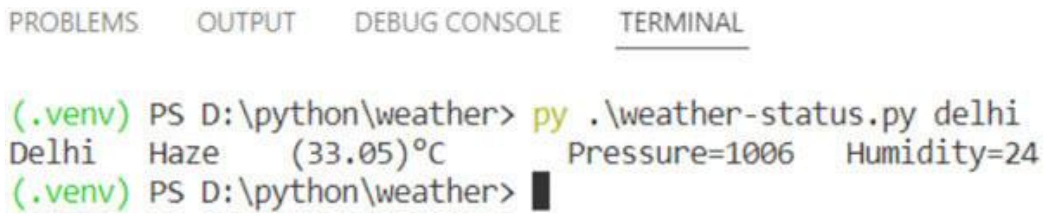
Code -part-1



```python

def acquire_weather_data(query_api_url):
    try:
        api_response = request.urlopen(query_api_url)
    except error.HTTPError as http_error:
        sys.exit(f"Error: {http_error.info}")

    weather_data = api_response.read()

    try:
        return json.loads(weather_data)
    except json.JSONDecodeError:
        sys.exit(",Unrecognized server response.")

def proces_and_output_weather_info(weather_data):
    city_name = weather_data["name"]
    weather_description = weather_data["weather"][0]["description"]
    temperature = weather_data["main"]["temp"]
    pressure = weather_data["main"]["pressure"]
    humidity = weather_data["main"]["humidity"]
    print(f"{city_name.capitalize()}\t"
              f"{weather_description.capitalize()}\t({temperature})°C\t"
              f"Pressure={pressure}\tHumidity={humidity}")

if __name__ == "__main__":
    args = read_args()
    api_query_url = make_api_url(args.city)
    weather_data = acquire_weather_data(api_query_url)
    proces_and_output_weather_info(weather_data)
```

Code -part-2

After completing the code, run the program and pass any valid city name as an argument and observe the output as shown in the following figure:



```
(.venv) PS D:\python\weather> py .\weather-status.py delhi
Delhi   Haze    (33.05)°C       Pressure=1006   Humidity=24
(.venv) PS D:\python\weather> 
```

Figure : *Weather Status Program Output*

The program is explained as follows:

•Line 45: Program starts by calling the function **read_args()** to get the city name.

•Line 8 – 15: Definition of **read_args()** function, it simply reads the city name passed as an argument while executing the program; if no argument is passed, then it will display help info. This is a standard way of argument handling.

Line 46: Function **make_api_url()** is called to form the full URL of the weather API.

•Line 17 – 19: Definition of function **make_api_url()**, it just creates a full API URL by concatenating all required pieces like base URL, city name, units, API key into a single string.

•Line 47: Function **acquire_weather_data()** is called to get the data.

•Line 21 – 32: Definition of function **acquire_weather_data()**. This is the main place where the API is called, and weather data is col- lected. We open API URL over HTTP using **urlopen()** function of Python's request module at line 23 and receives the response ob- ject. At line 27, we retrieve weather data from the response object and return this weather data in the form of JSON to the caller at line 30. Other lines are for exception handing.

•Line 48: Function **process_and_output_weather_info()** is called to display the weather info in a proper format.

•Line 34 – 42: Definition of **process_and_output_weather_info()** function. It receives the weather data JSON, retrieves all data points of interest like temperature, pressure, humidity, and so on, and then prints all data points in a particular human understandable format.

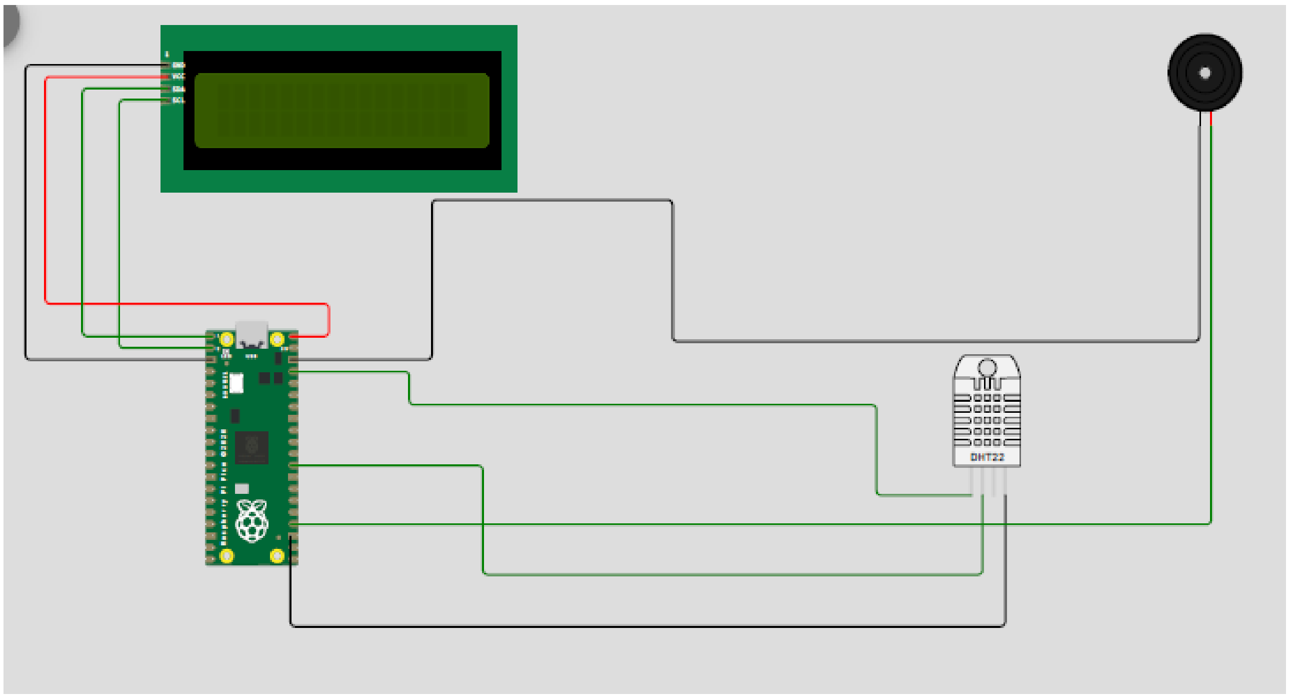
EXAMPLE 2: ACQUISITION AND PROCESSING DATA FROM TEMPERATURE SENSOR

The following code shows how to get data from DHT11 temperature/Humidity Sensor

```
from machine import Pin
from time import sleep
from dht import DHT22
from machine import PWM
from machine import I2C, Pin
from pico_i2c_lcd import I2cLcd


i2c = I2C(0, sda=Pin(0), scl=Pin(1), freq=400000)

I2C_ADDR = i2c.scan()[0]
```

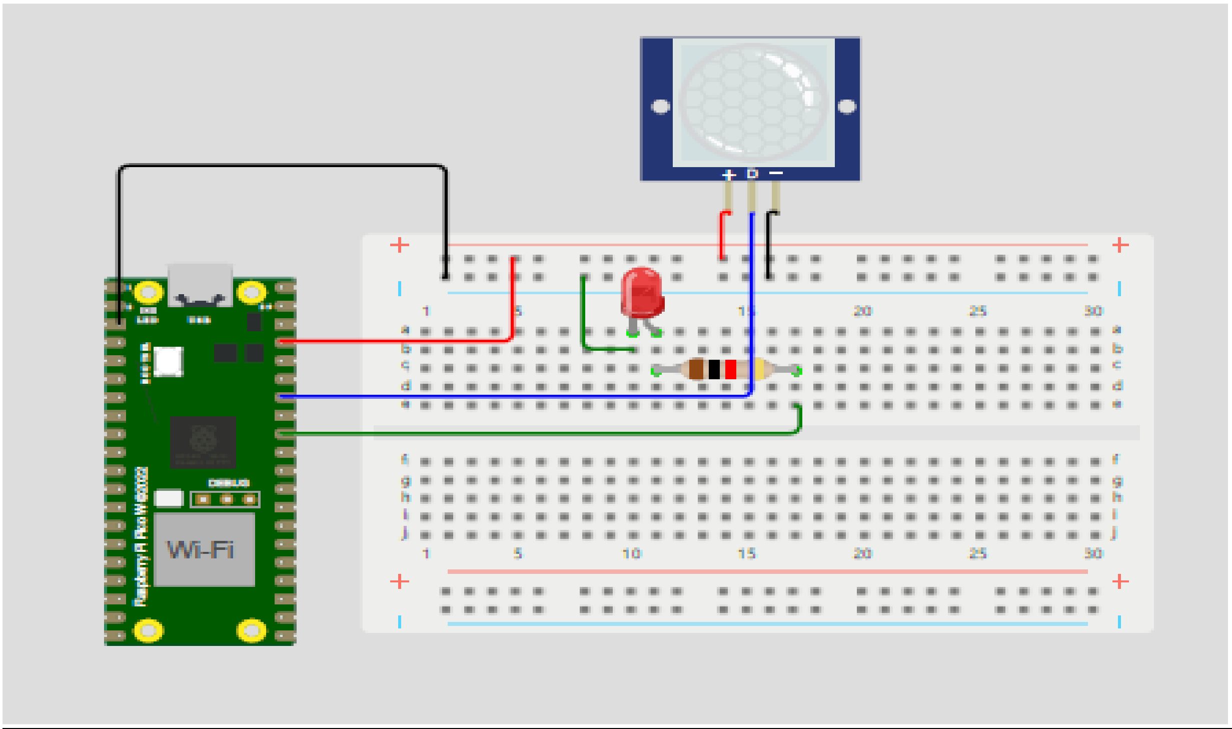
```
lcd = I2cLcd(i2c, I2C_ADDR, 2, 16)


dht = DHT22(Pin(22))
buzzer = PWM(Pin(18))

def bsound():
  buzzer.freq(900)
  buzzer.duty_u16(1000)
  sleep(1)
  buzzer.duty_u16(0)
  sleep(0.5)


while(True):
  dht.measure()
  temp = dht.temperature()
  hum = dht.humidity()

  print(f"temperature: {temp} *c Humidity{hum}")
  lcd.putstr(f"temp: {temp} *C \nHumi: {hum}%")
  sleep(3)
  lcd.clear()

  if(temp > 20 and hum > 30):
    bsound()


sleep(2)
```

However, the physical board and sensor needed to see this program in action and this can also be tested using virtualized boards and sensors for MicroPython.as shown in figure.



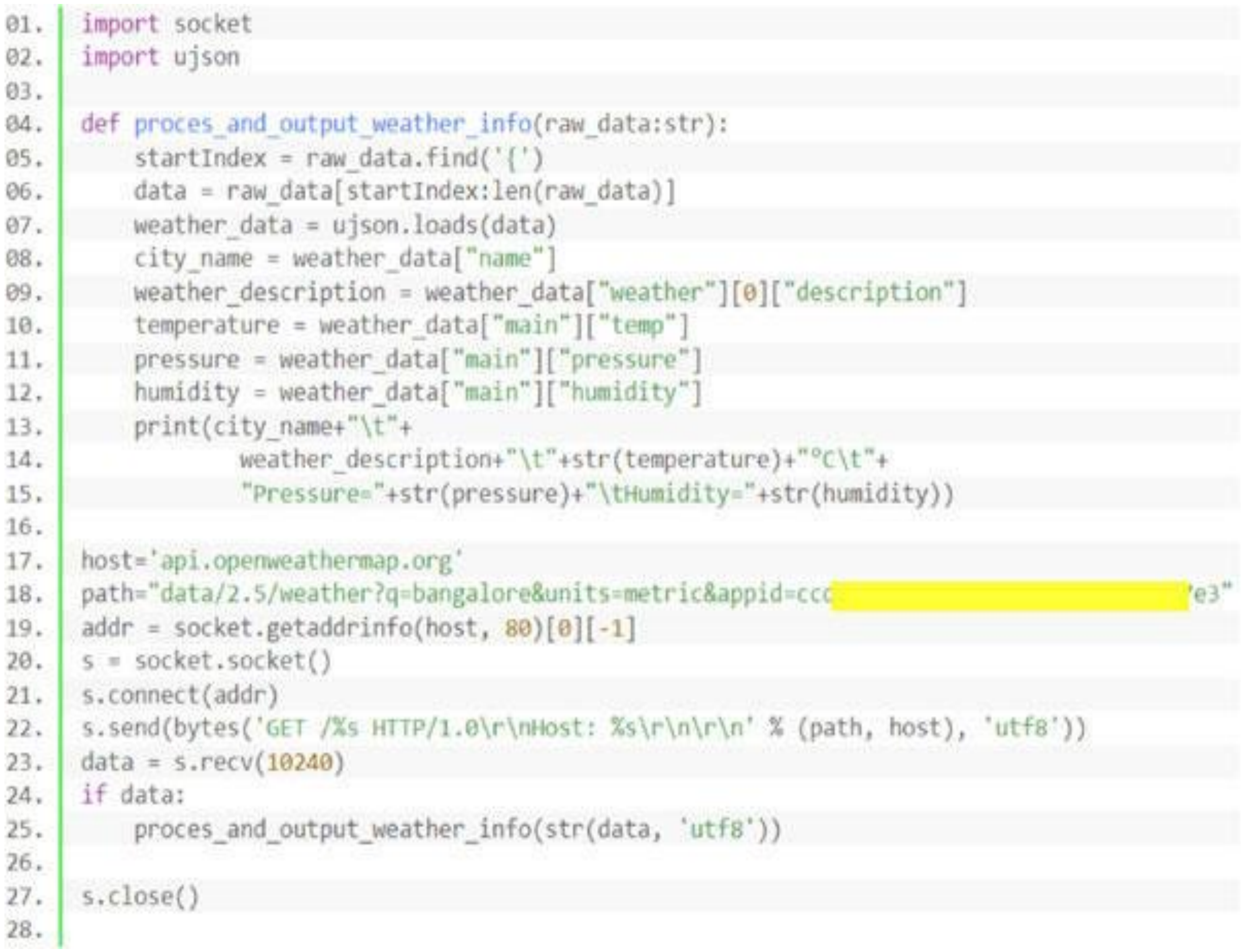
EXAMPLE-3: ACQUISITION AND PROCESSING DATA FROM MOTION SENSOR

```
from machine import Pin
from time import sleep
# Define the PIR sensor, LED
PIR_sensor = Pin(28, Pin.IN)
RedLED = Pin(27, Pin.OUT)

try:
  while True:
    # PIR sensor detects motion
    if PIR_sensor.value() == 1:
      print("\033c") # Clear Terminal
      RedLED.high()
      sleep(7)
      print("Motion Detected!")
  else:
    RedLED.low()
    sleep(1)

except keyboardInterrupt:
  print('Exiting')
  Pin.cleanup()
  print("Cleaning up!")
  display.clear()
```



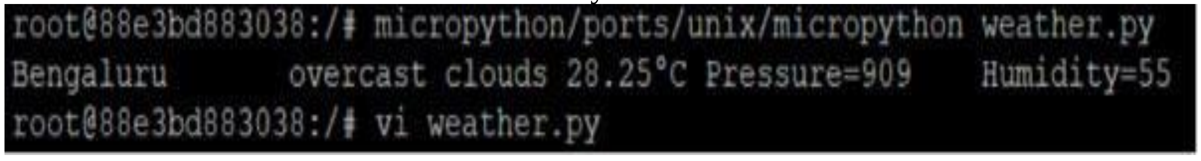
EXAMPLE-4: ACQUIRE DATA FROM HTTP API

The weather data program can also be written in MicroPython. For this we do not need physical board, here we can use a Unix version of MicroPython in Docker container environment . The following code can be used for getting weather data:



```
01. import socket
02. import ujson
03.
04. def proces_and_output_weather_info(raw_data:str):
05.     startIndex = raw_data.find('{')
06.     data = raw_data[startIndex:len(raw_data)]
07.     weather_data = ujson.loads(data)
08.     city_name = weather_data["name"]
09.     weather_description = weather_data["weather"][0]["description"]
10.     temperature = weather_data["main"]["temp"]
11.     pressure = weather_data["main"]["pressure"]
12.     humidity = weather_data["main"]["humidity"]
13.     print(city_name+"\t"+
14.             weather_description+"\t"+str(temperature)+"°C\t"+
15.             "Pressure="+str(pressure)+"\tHumidity="+str(humidity))
16.
17. host='api.openweathermap.org'
18. path="data/2.5/weather?q=bangalore&units=metric&appid=ccc[redacted]e3"
19. addr = socket.getaddrinfo(host, 80)[0][-1]
20. s = socket.socket()
21. s.connect(addr)
22. s.send(bytes('GET /%s HTTP/1.0\r\nHost: %s\r\n\r\n' % (path, host), 'utf8'))
23. data = s.recv(10240)
24. if data:
25.     proces_and_output_weather_info(str(data, 'utf8'))
26.
27. s.close()
28.
```

In the above code we are using socket module unlike Python's request module. Using socket, we connect to the API host at line 21, and send the HTTP **GET** request to the host with all parameters set in the path (note the city name **ban galore** is embedded) variable at line 22. After sending the re- quest, we call **recv()** function o f the socket to read the response with buffer size of 10240 bytes. Here we get the raw response which also contains all HTTP header info along with weather data hence a larger buffer is taken, otherwise we need to read until end in a loop. We convert the UTF-8 en- coded raw response bytes into string and pass to the **proces_and_out- put_weather_info ()** function. The **proces_and_output_weather_info()** func- tion is very similar to the one which we used in previous Python program. The only difference is we are extracting the weather data (which starts with **{** and goes till last byte) from the raw response data (which is a string) at line 5 and 6. The weather data is supposed to be in JSON format; hence we con- vert the extracted string into JSON at line 7, and the rest is just same. The following figure shows the output after running the code with MicroPython on Unix:



```
root@88e3bd883038:/# micropython/ports/unix/micropython weather.py
Bengaluru       overcast clouds 28.25°C Pressure=909    Humidity=55
root@88e3bd883038:/# vi weather.py
```