EXPERIMENT NUMBER : 4

EXPERIMENT NAME: RASPBERRYPI AND IoT CLOUD SERVER INTERFACE USING MQTT PROTOCOL 97

DATE : 10/11/2022 , THURSDAY

* AIM :

To analyze MQTT protocol used in the field of Smart devices and develop basic programming skills for deploying the protocol in hardware.

* INTEGRATED DEVELOPMENT ENVIRONMENT (IDE):

Name - Thonny 4.0.1
Publisher - Aivar Annamaa
Support line - https://thonny.org

(a) Data Transmission from Raspberry Pi to ThingSpeak IoT cloud Server :

→ Algorithm -

1. Import json, can be used to work with JSON (JavaScript object Notation) data.
2. Import random, implement pseudo-random number generators for various distributions.
3. Import requests, allows you to send HTTP/1.1 requests extremely easily.
4. Import threading, constructs higher-level threading interface on top of the lower level _thread module.
5. Import urllib.request, defines functions and classes which help in opening URLs (mostly HTTP) in a complex world.
6. Create a timer with interval and function as parameters.
7. Return an integer number selected element from the specified range.
8. Write a channel feed and API key.
9. Open the URL, which can be either a string or a Request object.

→ Python Code -

```
import urllib.request
import requests
import threading
import json
import random

def thingspeak_post():
    threading.Timer(15, thingspeak_post).start()
    val = random.randint(1, 30) # Alias for randrange(start, stop+1)
    URL = 'https://api.thingspeak.com/update?api_key=....'
    KEY = '.....xxxx.....'

    HEADER = '&field1={}&field2={}'.format(val, val)
    NEW_URL = URL + KEY + HEADER
    print(NEW_URL)

    data = urllib.request.urlopen(NEW_URL)
    print(data)

if _name_ == '_main_':
    thingspeak_post()
```

(b) Sensor Data Transmission from Raspberry Pi with SenseHAT to ThingSpeak IoT cloud Server:

→ Algorithm -

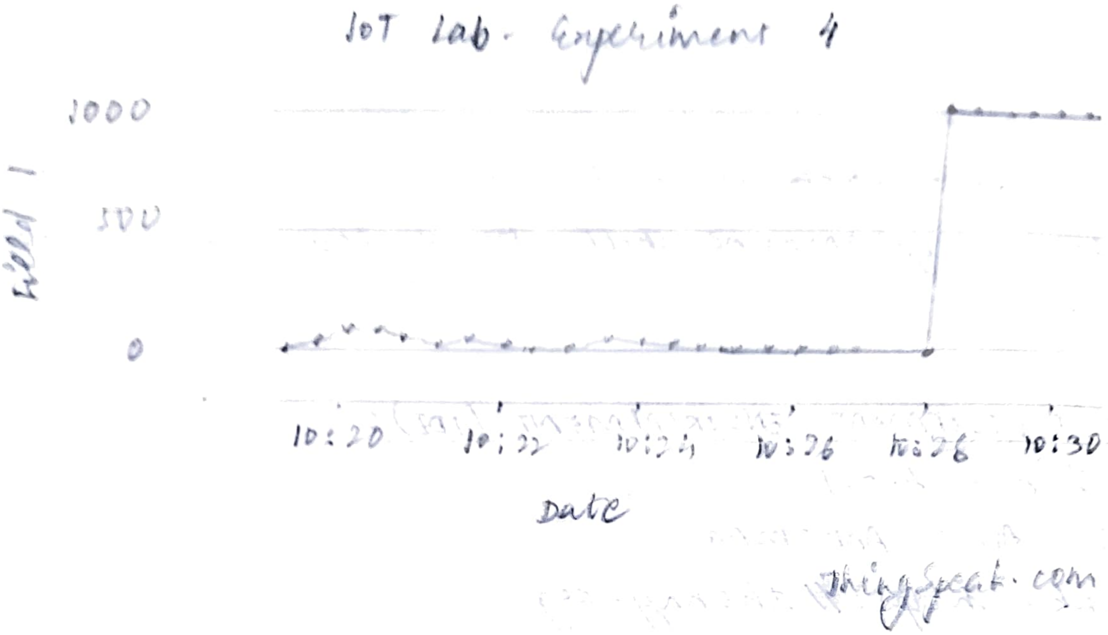
① Import json, can be used to work with JSON (JavaScript Object Notation) data.

② Import requests, allows you to send HTTP/1.1 requests extremely easily.

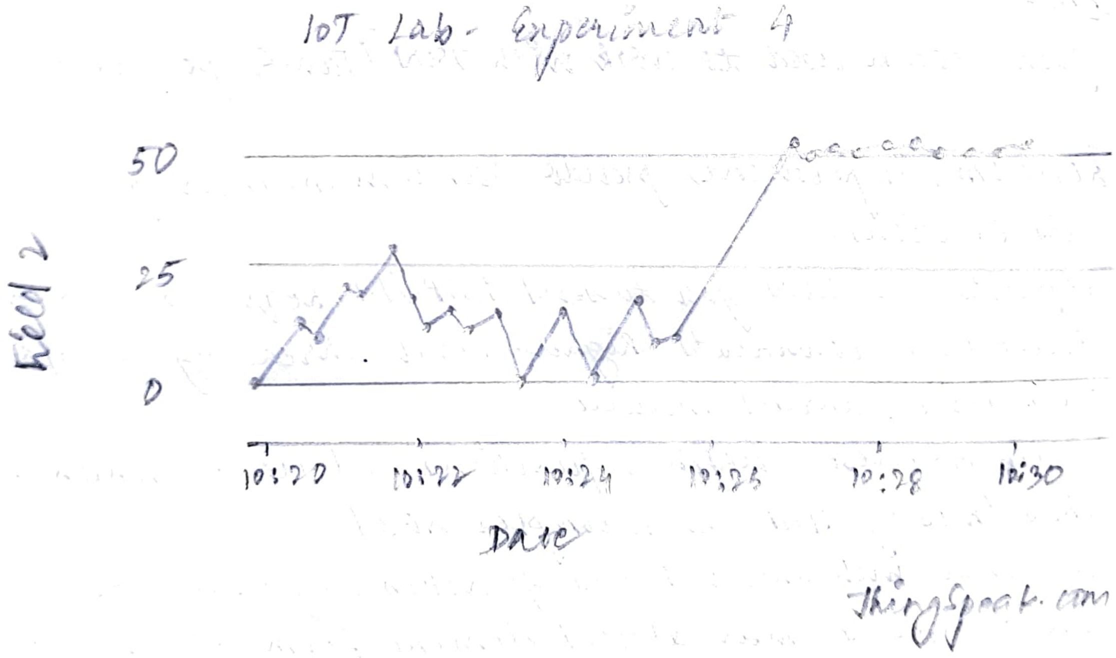
③ Import threading, constructs higher-level threading interfaces on top of the lower level_thread module.

* OUTPUTS:

Field 1 Chart



Field 2 Chart



④ Import urllib.request, defines functions and classes which help in opening URLs (mostly HTTP) in a complex world.

⑤ Import SenseHat from sense_hat, python module to control the Raspberry Pi Sense HAT. No arguments defaults to OFF.

⑥ Create a timer with interval and function as parameters.

⑦ Get the pressure and temperature values from the corresponding sensors.

→ Python Code -

```
import urllib.request
import requests
import threading
import json

from sense_hat import SenseHat
sense = SenseHat()
sense.clear()
sense.low_light = TRUE

def thingspeak_post():

    threading.Timer(15, thingspeak_post).start
    pressure = sense.get_pressure()
    humidity = sense.get_humidity()

    URL = 'https://api.thingspeak.com/update?api_key=...'
    KEY = '....xxxx....' # Write API Key
    HEADER = '&field1={}&field2={}'.format(pressure, humidity)

    NEW_URL = URL + KEY + HEADER
    print(NEW_URL)
    data = urllib.request.urlopen(NEW_URL)
    print(data)
```

```
if _name_ == '_main_':
    thingspeak_post ()
```

(c) **Sensor Data Transmission from Raspberry Pi with SenseHAT to Adafruit IoT Cloud Server:**

→ Algorithm -

① From sense-hat import SenseHat, python module to control the Raspberry Pi Sense HAT.

② No arguments defaults to OFF and toggle the LED matrix in low light mode, useful if the Sense HAT is being used in dark environment.

③ Import library and create instance of REST client.

④ Import time, provides time-related functions. Import random, implements pseudo-random number generators for various distributions.

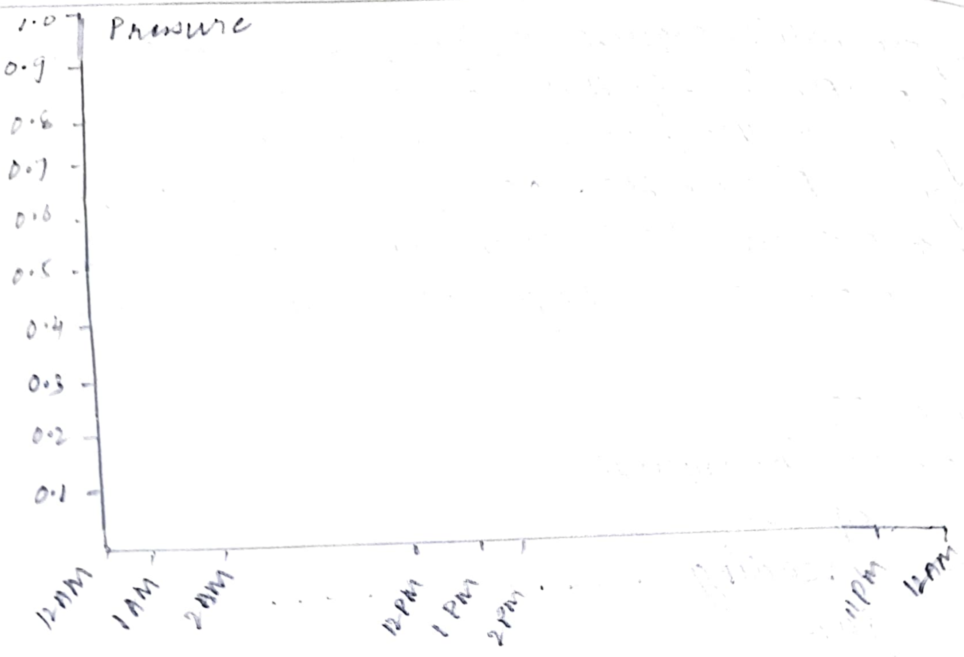
⑤ Get list of feeds:-
(i) 'Pressure' is IO Feed created in Adafruit.
(ii) 'humidity is IO Feed created in Adafruit.

⑥ Return an integer number selected element from the specified range.

⑦ If humidity and pressure values are not None, send pressure and humidity feeds to Adafruit IO.

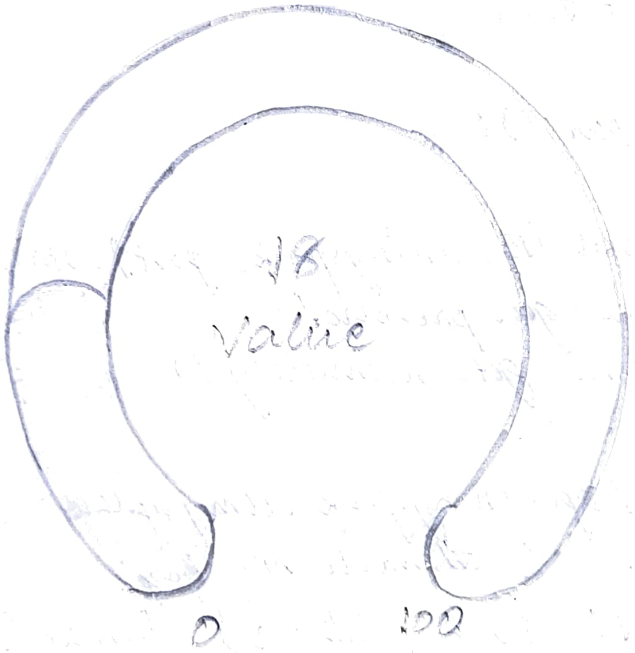
⑧ Else, print that 'Failed to get Pressure and Humidity Data from SenseHat!

⑨ Sleep (wait) for some time to avoid flooding Adafruit IO.



Pressure

Humidity



→ Python Code -

```
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()
sense.low_light = True

from Adafruit_IO import client, feed
import time
import random

READ_TIMEOUT = 5
ADAFRUIT_IO_KEY = '....xxxx....' # Active Key
ADAFRUIT_IO_USERNAME = '....xxxx....' # Username
aio = client(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

pressure_feed = aio.feeds('pressure')
humidity_feed = aio.feeds('humidity')

while True:

    val = random.randint(1,30) #Alias for randrange(start, stop+1)
    pressure = val
    humidity = val

    if humidity is not None and pressure is not None:
        print('Pressure = {0:0.1f} Pascal \n Humidity = {1:0.1f}%'
                .format(pressure, humidity))

        pressure = '%.2f' % (pressure)
        humidity = '%.2f' % (humidity)
        aio.send(pressure_feed.key, str(pressure))
        aio.send(humidity_feed.key, str(humidity))
    else:
        print ('Failed to get Pressure and Humidity Data from
SenseHat!')
    time. sleep (READ_ TIMEOUT)
```

(d) **Remote Device Control using Adafruit IOT Cloud Server on Raspberry Pi:**

→ Algorithm -

① From sense-hat import SenseHat, python module to control the Raspberry Pi Sense HAT.

② No argument defaults to OFF and toggle the LED matrix in low light mode, useful if the Sense HAT is being used in a dark environment.

③ Import sys, provides access to some variables used or maintained by the intergreter and to functions that interact strongly with the interpreter.

④ From Adafruit_IO import MQTTClient
- Username, Active key, key from Feeds

⑤ Define callback functions which will be called when certain events happen:-

(i) connected () - Will be called when the client connects
Subscribe to a feed

(ii) disconnected () - Will be called when the client disconnects optional argument arg can be an integer giving the exit or another type of object; Zero is considered "successful termination".

(iii) message () - Will be called when a subscribed feed has a new value. The feed_id parameter identifies the feed, and the payload parameter has the new value.

If payload = 1 ; print "Light ON"
Display a single text character on the LED mat

If payload = 0 ; print "Light OFF"
No arguments defaults to OFF

⑥ Open a new MQTT connection to the specified broker

Remote_Device_Control

→ Python Code-

```
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()
sense.low_light = True

import sys
from Adafruit_IO import MQTTClient

ADAFRUIT_IO_KEY = '....xxxx....' # Active key
ADAFRUIT_IO_USERNAME = 'S...xxxx....' # username
FEED_ID = '....xxx....' # key from Feeds

def connected (client):
    print ('Connected to Adafruit IO! Listening for {0} changes...'
format (FEED_ID))
    client.subscribe (FEED_ID)

def disconnected (client)
    print ('Disconnected from Adafruit IO!')
    sys.exit (1)

def mesage (client, feed_id, payload):
    print ('Feed {0} received new value : {1}.format (feed_id,
payload))

    if payload == "1":
      print ("Light ON")
      sense.show_letter ("O")

    if payload == "0":
      print ("Light OFF")
      sense.clear()
```

```
client = MQTT Client (ADAFRUIT_IO_USERNAME , ADAFRUIT_IO_KEY)
client.on_connect = connected
client.on_disconnect = disconnected
client.on_message = message
client.connect()
client.loop_blocking()
```

* RESULT:

Thus, analyzed MQTT protocols in the field of Smart devices and developed basic programming skills for deploying the protocol in hardware. All the simulation results were verified successfully.