IoT Experimental Learning
Week 2 Journal

1. **Describe the experience and what you hope to gain from participating in the experience.**
    - During this week I performed research and development in order to integrate a pub/sub broker and json into the existing project
    - I was also able to engage with other classmates through discussions to pass on information I gathered during this research by post two discussions on;
        1. Installing Mosquitto MQTT broker on Raspberry Pi 4.
        2. Utilization of a json library "jsonpickle" as an alternative to the baked in python json module.

2. **Provide an overview of tasks and key activities (training, discussions, labs, assessments, etc.) in which you were engaged during the week.**

    For week 2 I accomplished the following tasks in chronological order;

❖ **Monday January 27, 2020**, completed reviewing this week's assignment for the project.
- ➢ I performed an internet search and read though the MQTT Specification at https://mqtt.org/
- ➢ Completed reviewing the Mosquitto MQTT Broker implementation at https://mosquitto.org/
- ➢ Researched json library alternatives for python, decided on jsonpickle located at https://jsonpickle.github.io/
- ➢ Created Class Discussion post for Mosquitto install on Raspberry Pi 4 with Buster

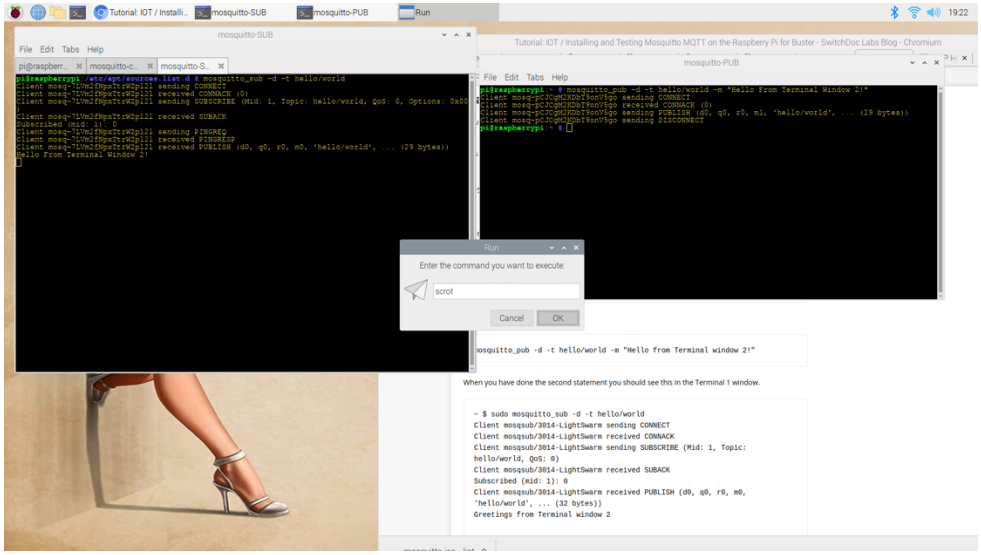


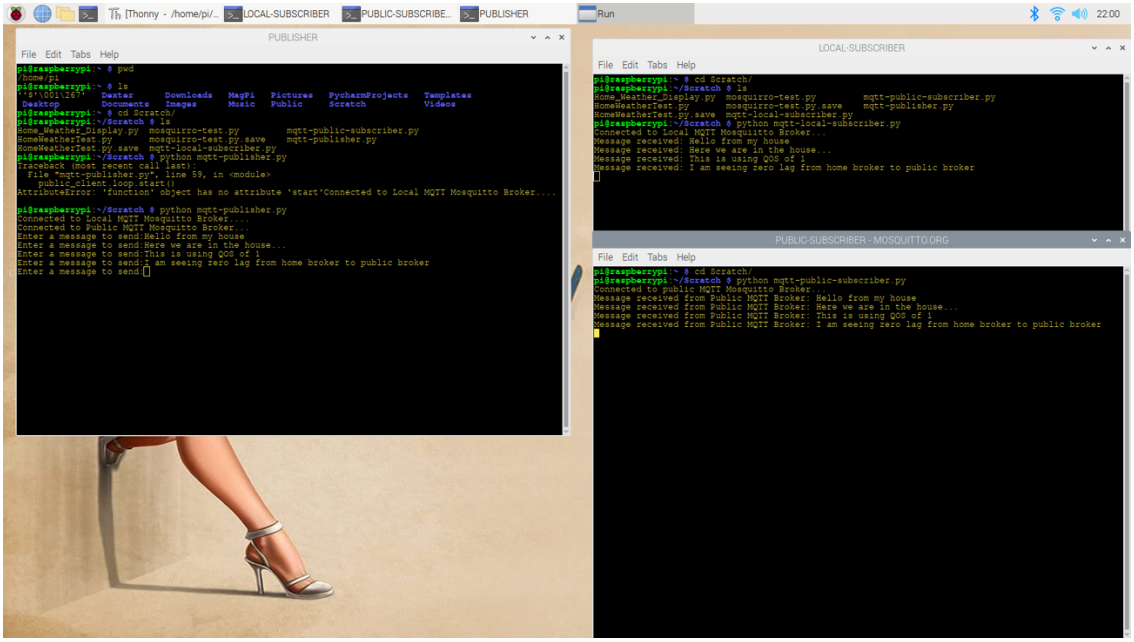
Figure 1Mosquitto Install Test

- **Tuesday January 21, 2020**, complete review of jsonpickle library API and specification.
  - I created new class discussion post to inform class of my findings.

- **Thursday January 30, 2020**, completed required weekly Class Discussion post

- **Saturday February 01, 2020**,

  - Created Mosquitto Pub/Sub test with local and public broker



  - Created jsonpickle test code and executed as stand alone test
  -

```
import jsonpickle
import simple_message_module as sm
from datetime import datetime

now = datetime.now()
#timestamp = now.timestamp()

smObj = sm.SimpleMessage(msg="A Test for Json encoding",
ts=now.strftime("%m/%d/%Y %H:%M:%S"))
smObj.printmessage()

smJson = jsonpickle.encode(value=smObj,
unpicklable=False)
print(smJson)
```

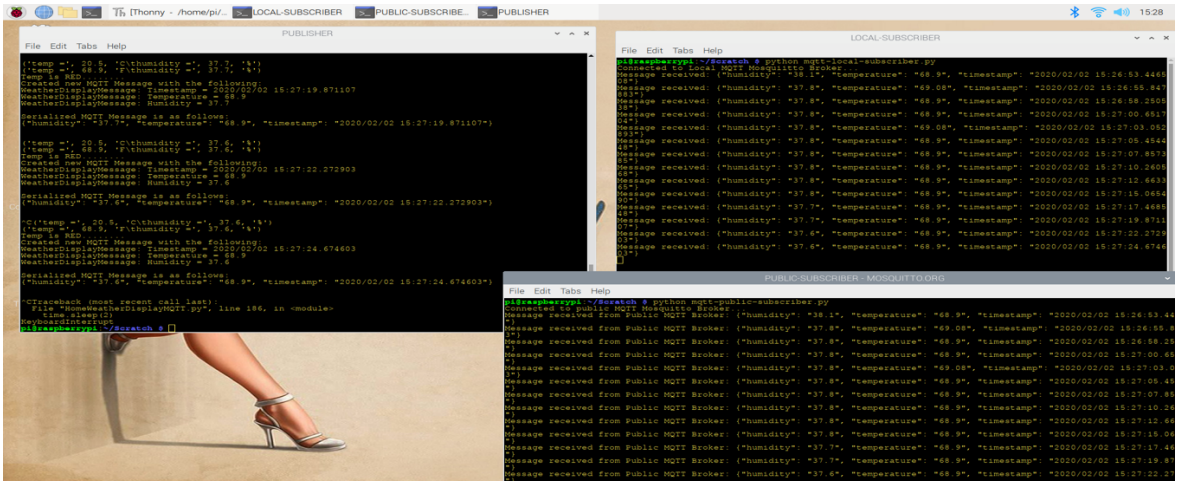
```
# Deserialize the json to a python obj
obj = jsonpickle.decode(smJson)

# Get the message from the obj using the schema key of message
msg = obj.get("message")

# print the retrieved message to the console
print("The deserialized message is " +  "\"" + msg + "\"")
```

❖

```
import datetime


class SimpleMessage:

    def __init__(self, msg, ts):
        self.timestamp = ts
        self.message = msg


    def printmessage(self):
        print("TimeStamp: " + self.timestamp)
        print("Message: " + self.message)
```

❖ Sunday February 02, 2020, Completed integration and testing of Mosquitto and jsonpickle into existing Project week 1 code base.

- ➢ Completed code testing with localhost and public broker at test.mosquitto.org
- ➢ I created encapsulation of publisher in a python class, but I did not have a chance to introduce into code base this week.
- ➢ Plan on integration new class in week 3



HomeWeatherDisplayMQTT.py
This is the running program illustrated above, using the message encapsulation class module

```
# Home_Weather_Display.py

from grovepi import *
from grove_rgb_lcd import *
import time
from math import isnan
import paho.mqtt.client as mqttClient
import weather_display_message as wdm


def local_on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to Local MQTT Mosquitto Broker....")
        global LOCAL_CONNECTED  # Use global variable
        LOCAL_CONNECTED = True  # Signal connection
    else:
        print("Connection to Local MQTT Mosquitto Broker failed..." +rc)


def public_on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to Public MQTT Mosquitto Broker...")
        global PUBLIC_CONNECTED  # Use global variable
        PUBLIC_CONNECTED = True  # Signal connection
    else:
        print("Connection to Public MQTT Mosquitto Broker failed..." +rc)


def local_on_publish(client, userdata, mid):
    print("LOCAL CLIENT PUBLISHED MESSAGE WITH ID: " + mid)


def public_on_publish(client, userdata, mid):
    print("PUBLIC CLIENT PUBLISHED MESSAGE WITH ID: " + mid)
    mqtt_client.publish(topic=self.topic, payload=mqtt_msg.serialize(), qos=1)


# connect the DHt sensor to port 7
dht_sensor_port = 7
# use 0 for the blue-colored sensor and 1 for the white-colored sensor
dht_sensor_type = 1
# set the default back lighting to BLUE
```

```
setRGB(0, 0, 255)
# clear the LED Screen
setText_norefresh(" ")


# global variables for connection state and other data
LOCAL_CONNECTED = False
PUBLIC_CONNECTED = False
PUBLISH_TOPIC = "SNHU/IT697/john_richardson3/sensor/data/temphum/"

local_broker_address = "localhost"
public_broker_address = "test.mosquitto.org"
public_broker_port = 1883
# user = "yourUser"
# password = "yourPassword"

# create the MQTT Clients
local_client = mqttClient.Client("LOCAL")  # create new instance
public_client = mqttClient.Client("PUBLIC")

# local_client.username_pw_set(user, password=password)  # set username and password

# attach functions to callbacks
local_client.on_connect = local_on_connect
local_client.on_publish = local_on_publish
public_client.on_connect = public_on_connect
public_client.on_publish = public_on_publish

local_client.connect(local_broker_address)  # connect to broker
public_client.connect(public_broker_address, port=public_broker_port)

# start the connection loops
local_client.loop_start()
public_client.loop_start()

# Wait for connections
while not LOCAL_CONNECTED and not PUBLIC_CONNECTED:
    time.sleep(0.1)

while True:
    try:
        # get the temperature and Humidity from the DHT sensor
        [ temp,hum ] = dht(dht_sensor_port,dht_sensor_type)
        print("temp =", temp, "C\thumidity =", hum,"%")

        # check if we have nans
```

```
    # if so, then raise a type error exception
    if isnan(temp) is True or isnan(hum) is True:
        raise TypeError('nan error')

# Convert the aquired temperature to Fahrenheit
    tempf = (temp * 1.8) + 32

# Print the converted temperature and humidity
# to the shell/console
    print('temp =', tempf, 'F\thumidity =', hum, '%')

# If the temperature in Fahrenheit is greater then 60
# change the LeD back lighting to RED
# and print this condition
    if tempf > 68:
        print('Temp is RED.....')
        setRGB(255,0,0)
        pass
    else:
    # If its not greater then 68
    # set the backlighting to BLUE and
    # print color condition to console
        print('Temp is BLUE')
        setRGB(0,255.0)

    t = str(temp)
    h = str(hum)
    tf = str(tempf)

    # setText_norefresh("Temp:" + tf + "F\n" + "Humidity:" + h + "%")

    mqtt_msg = wdm.WeatherDisplayMessage(temp=tf, humidity=h)
    print("Created new MQTT Message with the following:")
    mqtt_msg.print_raw_content()
    print("")
    print("Serialized MQTT Message is as follows:")
    mqtt_msg.print_serialized_obj()
    print("")
    print("")

    local_client.publish(topic=PUBLISH_TOPIC, payload=mqtt_msg.serialize(), qos=1)
    public_client.publish(topic=PUBLISH_TOPIC, payload=mqtt_msg.serialize(), qos=1)

# allowing the screen to refresh on writes
    setText("Temp:" + tf + "F\n" + "Humidity:" + h + "%")
```

```
    except (IOError, TypeError) as e:
        print(str(e))
        setText_norefresh(" ")

        # and since we got a type error
        # then reset the LCD's text

    except KeyboardInterrupt as e:
        print(str(e))

        setText_norefresh(" ")
        # since we're exiting the program
        # it's better to leave the LCD with a blank text
        break

    # wait some time before re-updating the LCD
    time.sleep(2)
```

This is the message encapsulation and obfuscates the serialization of the json payload

weather_display_message.py

```
from datetime import datetime
import jsonpickle


class WeatherDisplayMessage:

    def __init__(self, temp, humidity):
        now = datetime.now()
        self.temperature = temp
        self.humidity = humidity
        self.timestamp = now.strftime("%Y/%m/%d %H:%M:%S.%f")

    def serialize(self):
        return jsonpickle.encode(self, unpicklable=False)

    def print_raw_content(self):
        print("WeatherDisplayMessage: Timestamp = " + self.timestamp)
        print("WeatherDisplayMessage: Temperature = " + self.temperature)
        print("WeatherDisplayMessage: Humidity = " + self.humidity)

    def print_serialized_obj(self):
        print(self.serialize())
```

This is the class that I would like to introduce in week 3, it encapsulates the publishing and makes the code cleaner.

HomeWeatherDisplayPublisher.py

```
import paho.mqtt.client as mqttClient
import weather_display_message as wdm


class HomeWeatherPublisher:
    CONNECTED = None  # type: bool

    def __init__(self, mqtt_host, port, mqtt_client_id, topic):
        self.mqtt_host = mqtt_host
        self.port = port
        self.topic = topic
        self.CONNECTED = False
        self.mqtt_client = mqttClient.Client(mqtt_client_id)
        self.mqtt_client.on_connect = self.on_connect

    def on_connect(self, client, userdata, flags, rc):
        if rc == 0:
            print("Connected to " + userdata + " MQTT Mosquitto Broker...")
            # PUBLIC_CONNECTED  # Use global variable
            self.CONNECTED = True  # Signal connection
        else:
            print("Connection to" + userdata + " MQTT Mosquitto Broker failed..." + rc)
            self.stop_publishing()

    def connect(self):
        self.mqtt_client.connect(host=self.mqtt_host, port=self.port)
        self.mqtt_client.loop_start()

    def stop_publishing(self):
        self.mqtt_client.disconnect()
        self.mqtt_client.loop_stop()

    def publish_msg(self, temp, humidity):
        mqtt_msg = wdm.WeatherDisplayMessage(temp=temp, humidity=humidity)
        print("Created new MQTT Message with the following:")
        mqtt_msg.print_raw_content()
        print("")
        print("Serialized MQTT Message is as follows:")
        mqtt_msg.print_serialized_obj()
        print("")
```

```
    print("")
    self.mqtt_client.publish(topic=self.topic, payload=mqtt_msg.serialize(), qos=1)

def is_connect(self):
    return self.CONNECTED

def get_client(self):
    return self.mqtt_client

def get_topic(self):
    return self.topic
```