Homework Assigned:

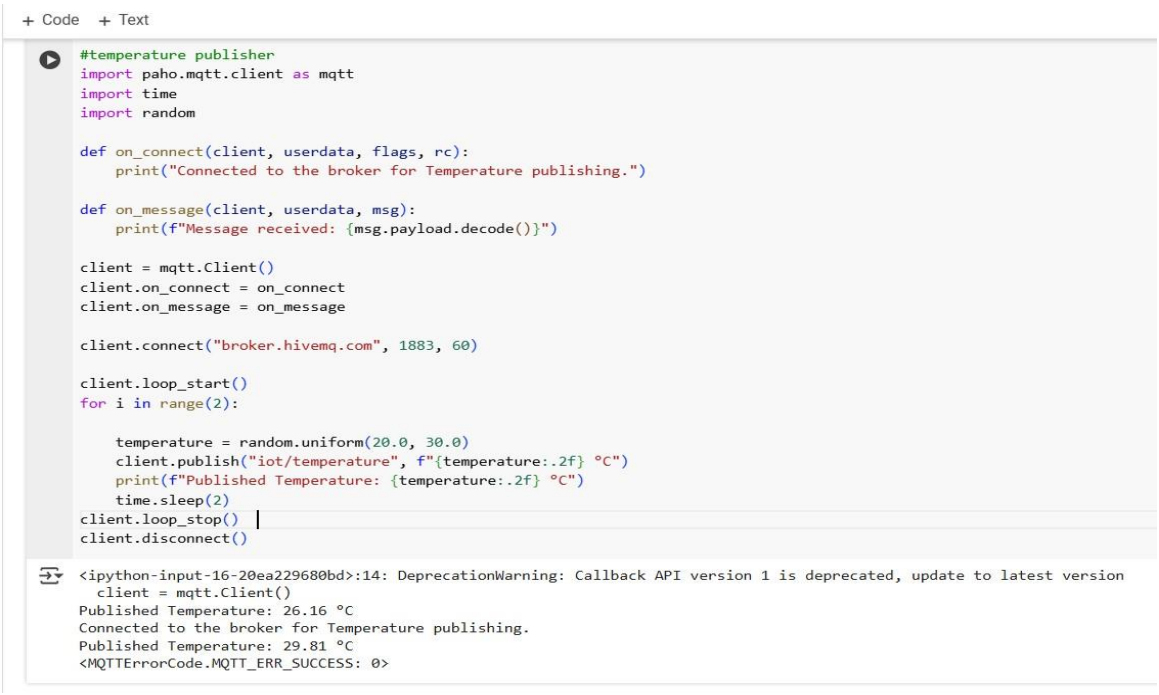
AIM: To simulate IoT device communication using MQTT or HTTP protocols.

Temperature:

1) A Temperature Publisher that publishes random temperature readings.

Program code:

```
import paho.mqtt.client as mqtt import
time import random
def on_connect(client, userdata,flags,rc):
    print("Connected to the broker for Temperature publishing.")
def on_message(client, userdata, msg):
    print(f"Message received: {msg.payload.decode()}")
client = mqtt.Client()
client.on_connect = on_connect
client.on_message=on_message
client.connect("broker.hivemq.com",1883,60)
client.loop_start()
for i in range(2):
    temperature = random.uniform(20.0, 30.0)
    client.publish("iot/temperature",f"{temperature:.2f}°C")
    print(f"Published Temperature: {temperature:.2f} °C")
time.sleep(2)
client.loop_stop()
client.disconnect()
```



```
+ Code + Text

#temperature publisher
import paho.mqtt.client as mqtt
import time
import random

def on_connect(client, userdata, flags, rc):
    print("Connected to the broker for Temperature publishing.")

def on_message(client, userdata, msg):
    print(f"Message received: {msg.payload.decode()}")

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("broker.hivemq.com", 1883, 60)

client.loop_start()
for i in range(2):

    temperature = random.uniform(20.0, 30.0)
    client.publish("iot/temperature", f"{temperature:.2f} °C")
    print(f"Published Temperature: {temperature:.2f} °C")
    time.sleep(2)
client.loop_stop()
client.disconnect()

<ipython-input-16-20ea229680bd>:14: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
  client = mqtt.Client()
Published Temperature: 26.16 °C
Connected to the broker for Temperature publishing.
Published Temperature: 29.81 °C
<MQTTErrorCode.MQTT_ERR_SUCCESS: 0>
```

Explanation and logic of Code

This Python code demonstrates how to use the Paho-MQTT library to connect to an MQTT broker, subscribe to a topic, and publish a message to that topic. Let's go through the code line by line:

1. **Importing Libraries:**
   - `paho.mqtt.client`: This is the Paho MQTT client library, which allows you to create an MQTT client for publishing and subscribing to messages.
   - `time`: This library is used to introduce delays in the code (e.g., to wait between publishing messages).
   - `random`: This library is used to generate random temperature values for simulation.
2. **Defining the `on_connect` Callback:**
   - This function is called when the client successfully connects to the MQTT broker.
   - It prints a message indicating that the connection was successful.
3. **Defining the `on_message` Callback:**
   - This function is called when a message is received on a subscribed topic.

- It decodes the message payload and prints it. In this code, it is defined but not used since the client is only publishing messages.

4. **Creating the MQTT Client:**
   - `client = mqtt.Client()`: This creates a new instance of the MQTT client.
   - The `on_connect` and `on_message` callback functions are assigned to the client. This means that when the client connects to the broker, the `on_connect` function will be called, and when a message is received, the `on_message` function will be called.
5. **Connecting to the MQTT Broker:**
   - This line connects the client to the specified MQTT broker (`broker.hivemq.com`) on port 1883. The last parameter (60) is the keep-alive interval in seconds, which is used to maintain the connection.
6. **Starting the Loop:**
   - This starts a background thread that handles network traffic and dispatches callbacks. It allows the client to process incoming messages while the main thread continues executing.
7. **Publishing Temperature Data:**
   - This loop runs twice (as specified by `range(2)`).
   - Inside the loop:
     i. `temperature = random.uniform(20.0, 30.0)`: This generates a random temperature value between 20.0 and 30.0 degrees Celsius.
     ii. `client.publish("iot/temperature", f"{temperature:.2f} °C")`: This publishes the temperature value to the `iot/temperature` topic, formatted to two decimal places.
     iii. `print(f"Published Temperature: {temperature:.2f} °C")`: This prints the published temperature to the console.
     iv. `time.sleep(2)`: This introduces a 2-second delay before the next iteration of the loop.
8. **Stopping the Loop and Disconnecting:**
   - `client.loop_stop()`: This stops the background loop that processes network traffic.

- `client.disconnect()`: This disconnects the client from the MQTT broker, closing the connection cleanly.

Logic:

This code sets up an MQTT client that connects to a broker and publishes random temperature readings to the `iot/temperature` topic. It does this twice, with a 2-second interval between each publication. The `on_message` callback is defined but not utilized in this code since it is only publishing messages. The code demonstrates a simple use case of the publish-subscribe mechanism in MQTT.

Observation Table:

Published Message

```
Connected to the broker  for Temperature publishing.
Published Temperature: 29.81 °C
```

2) A Temperature Subscriber that listens for temperature messages.

Program code:

```
import paho.mqtt.client as mqtt
def on_connect(client, userdata,flags,rc):
    print("Connected to the broker for Temperature subscribing.")
    client.subscribe("iot/temperature")

def on_message(client, userdata, msg):
    print(f"Message received on topic {msg.topic}: {msg.payload.decode()}")

client=mqtt.Client()
client.on_connect=on_connect
client.on_message = on_message
client.connect("broker.hivemq.com",1883,60)
client.loop_start()
```

print("Listening for temperature messages...")
client.loop_stop()
client.disconnect()

```python
#temperature
import paho.mqtt.client as mqtt

# Define callback functions
def on_connect(client, userdata, flags, rc):
    print("Connected to the broker for Temperature subscribing.")
    client.subscribe("iot/temperature")

def on_message(client, userdata, msg):
    print(f"Message received on topic {msg.topic}: {msg.payload.decode()}")

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect("broker.hivemq.com", 1883, 60)
client.loop_start()

print("Listening for temperature messages...")
client.loop_stop()
client.disconnect()
```

```
<ipython-input-52-42f92727af2e>:12: DeprecationWarning: Callback API version 1 i
  client = mqtt.Client()
Listening for temperature messages...
Message received on topic iot/temperature: 26.16 °C
Connected to the broker for Temperature subscribing.
<MQTTErrorCode.MQTT_ERR_SUCCESS: 0>
```

Explanation and logic of Code

This Python code demonstrates how to use the Paho-MQTT library to connect to an MQTT broker, subscribe to a topic, and publish a message to that topic. Let's go through the code line by line:

1. **Importing the Paho MQTT Library:**
   - This line imports the Paho MQTT client library, which is used to create an MQTT client for subscribing to messages.
2. **Defining the `on_connect` Callback:**
   - This function is called when the client successfully connects to the MQTT broker.
   - It takes four parameters:
     i. `client`: The client instance for this callback. ii. `userdata`: User-defined data (not used here).
     iii. `flags`: Response flags from the broker (not used here).

iv. `rc`: Connection result code (0 means success).

- Inside the function:
  i. It prints a message indicating that the connection was successful.
  ii. It subscribes to the `iot/temperature` topic, meaning the client will start receiving messages published to this topic.

3. Defining the `on_message` Callback:

- This function is called when a message is received on a subscribed topic.
- It takes three parameters:
  i. `client`: The client instance for this callback. ii. `userdata`: User-defined data (not used here).
  iii. `msg`: The message object containing the topic and payload.
- Inside the function:
  i. It decodes the message payload and prints it along with the topic from which it was received. This allows the subscriber to see the content of the message.

4. Assigning Callback Functions:

- The `on_connect` function is assigned to handle connection events.
- The `on_message` function is assigned to handle incoming messages. This means that when a message is received on the subscribed topic, the `on_message` function will be called.

5. Connecting to the MQTT Broker:

- This line connects the client to the specified MQTT broker (`broker.hivemq.com`)
  on port 1883.
- The last parameter (60) is the keep-alive interval in seconds, which is used to maintain the connection. If the client does not send any messages within this interval, it will send a ping to the broker to keep the connection alive.

Logic:

This code sets up an MQTT subscriber that connects to a broker and listens for messages on the `iot/temperature` topic. However, there is a logical issue: the `client.loop_stop()` is called immediately after starting the loop, which means the subscriber will not have time to receive any messages. To fix this, you would typically want to use `client.loop_forever()` instead of `client.loop_start()` and `client.loop_stop()` to keep the subscriber running indefinitely until interrupted.

Observation Table:

Published message

```
Connected to the broker for Temperature publishing.
Published Temperature: 29.81 °C
```

Received Message

```
Listening for temperature messages...
Message received on topic iot/temperature: 26.16 °C
Connected to the broker for Temperature subscribing.
```

Pressure:

1. A Pressure Publisher that publishes random pressure readings.

Program code:

```
import paho.mqtt.client as mqtt
import time
import random
def on_connect(client, userdata,flags,rc):
    print("Connected to the broker for Pressure publishing.")
def on_message(client, userdata, msg):
    print(f"Message received: {msg.payload.decode()}")
client = mqtt.Client()
client.on_connect = on_connect
```

```
client.on_message = on_message
client.connect("broker.hivemq.com",1883,60)
client.loop_start()
for i in range(5):
    pressure = random.uniform(950.0, 1050.0)
    client.publish("iot/pressure", f"{pressure:.2f} hPa")
    print(f"Published Pressure: {pressure:.2f} hPa")
    time.sleep(2)
client.loop_stop()
client.disconnect()
```

```
import time
import random
def on_connect(client, userdata, flags, rc):
    print("Connected to the broker for Pressure publishing.")

def on_message(client, userdata, msg):
    print(f"Message received: {msg.payload.decode()}")

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("broker.hivemq.com", 1883, 60)
client.loop_start()

for i in range(5):

        pressure = random.uniform(950.0, 1050.0)
        client.publish("iot/pressure", f"{pressure:.2f} hPa")
        print(f"Published Pressure: {pressure:.2f} hPa")
        time.sleep(2)

client.loop_stop()
client.disconnect()
```

```
<ipython-input-12-5d822247cc37>:10: DeprecationWarning: Callback
  client = mqtt.Client()
Published Pressure: 992.08 hPa
Connected to the broker for Pressure publishing.
Published Pressure: 989.65 hPa
```

Explanation and logic of Code

This Python code demonstrates how to use the Paho-MQTT library to connect to an MQTT broker, subscribe to a topic, and publish a message to that topic. Let's go through the code line by line:

1. **Importing the Paho MQTT Library:**
   - This line imports the Paho MQTT client library, which is used to create an MQTT client for subscribing to messages.
2. **Defining the `on_connect` Callback:**
   - This function is called when the client successfully connects to the MQTT broker.
   - It takes four parameters:
     i. `client`: The client instance for this callback. ii. `userdata`: User-defined data (not used here).
     iii. `flags`: Response flags from the broker (not used here).
     iv. `rc`: Connection result code (0 means success).
   - Inside the function:
     i. It prints a message indicating that the connection was successful.
     ii. It subscribes to the `iot/temperature` topic, meaning the client will start
     receiving messages published to this topic.
3. **Defining the `on_message` Callback:**
   - This function is called when a message is received on a subscribed topic.
   - It takes three parameters:
     i. `client`: The client instance for this callback. ii. `userdata`: User-defined data (not used here).
     iii. `msg`: The message object containing the topic and payload.
   - Inside the function:
     i. It decodes the message payload and prints it along with the topic from which it was received. This allows the subscriber to see the content of the message.
4. **Assigning Callback Functions:**

- The `on_connect` function is assigned to handle connection events.
- The `on_message` function is assigned to handle incoming messages. This means that when a message is received on the subscribed topic, the `on_message` function will be called.

5. **Connecting to the MQTT Broker:**
   - This line connects the client to the specified MQTT broker (`broker.hivemq.com`) on port 1883.
   - The last parameter (60) is the keep-alive interval in seconds, which is used to maintain the connection. If the client does not send any messages within this interval, it will send a ping to the broker to keep the connection alive.
6. **Stopping the Loop:**
   - This line stops the background loop that processes network traffic. However, it is important to note that this line will stop the loop immediately after starting it, which means the subscriber will not actually receive any messages. This line should typically be placed after a mechanism to keep the subscriber running (like a `try` **block with** `loop_forever()`**).**

Observation Table:

Published message

```
Connected to the broker for Pressure publishing.
Published Pressure: 989.65 hPa
```

2) A Pressure Subscriber that listens for pressure messages.

Program code:

```
import paho.mqtt.client as mqtt
def on_connect(client, userdata, flags, rc):
    print("Connected to the broker for Pressure subscribing.")
    client.subscribe("iot/pressure")
def on_message(client, userdata, msg):
    print(f"Message received on topic {msg.topic}: {msg.payload.decode()}")
```

```
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect("broker.hivemq.com",1883,60)
client.loop_start()
print("Listening for pressure messages...")
client.loop_forever()
client.loop_stop()
client.disconnect()
```

```
import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc):
    print("Connected to the broker for Pressure subscribing.")
    client.subscribe("iot/pressure")

def on_message(client, userdata, msg):
    print(f"Message received on topic {msg.topic}: {msg.payload.decode()}")

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("broker.hivemq.com", 1883, 60)
client.loop_start()

print("Listening for pressure messages...")
client.loop_forever()
client.loop_stop()
client.disconnect()
```

```
<ipython-input-8-080d5ab0b012>:10: DeprecationWarning: Callback API version 1 i
  client = mqtt.Client()
Message received on topic iot/pressure: 996.89 hPa
Message received on topic iot/pressure: 996.89 hPa
Listening for pressure messages...
Message received on topic iot/pressure: 996.89 hPa
Connected to the broker for Pressure subscribing.
```

Explanation and logic of Code

This Python code demonstrates how to use the Paho-MQTT library to connect to an MQTT broker, subscribe to a topic, and publish a message to that topic. Let's go through the code line by line:

1. **Importing the Paho MQTT Library:**
   - This line imports the Paho MQTT client library, which is used to create an MQTT client for subscribing to messages.
2. **Defining the `on_connect` Callback:**
   - This function is called when the client successfully connects to the MQTT broker.

It takes four parameters:

i. `client`: The client instance for this callback.

ii. `userdata`: User-defined data (not used here).

iii. `flags`: Response flags from the broker (not used here).

iv. `rc`: Connection result code (0 means success).

- Inside the function:
  i. It prints a message indicating that the connection was successful.
  ii. It subscribes to the `iot/pressure` topic, meaning the client will start receiving messages published to this topic.

3. **Defining the `on_message` Callback:**
   - This function is called when a message is received on a subscribed topic.
   - It takes three parameters:
     i. `client`: The client instance for this callback.
     ii. `userdata`: User-defined data (not used here).
     iii. `msg`: The message object containing the topic and payload.
4. This line connects the client to the specified MQTT broker (`broker.hivemq.com`) on port 1883.
5. The last parameter (60) is the keep-alive interval in seconds, which is used to maintain the connection. If the client does not send any messages within this interval, it will send a ping to the broker to keep the connection alive.

Observation Table:

Published message

```
Connected to the broker for Pressure publishing.
Published Pressure: 996.89 hPa
```

Received Message

```
Listening for pressure messages...
Message received on topic iot/pressure: 996.89 hPa
Connected to the broker for Pressure subscribing.
Connected to the broker for Pressure subscribing.
Message received on topic iot/pressure: 1043.04 hPa
```

Conclusion : We have learned to set up basic MQTT communication, publish messages, and subscribe to topics.