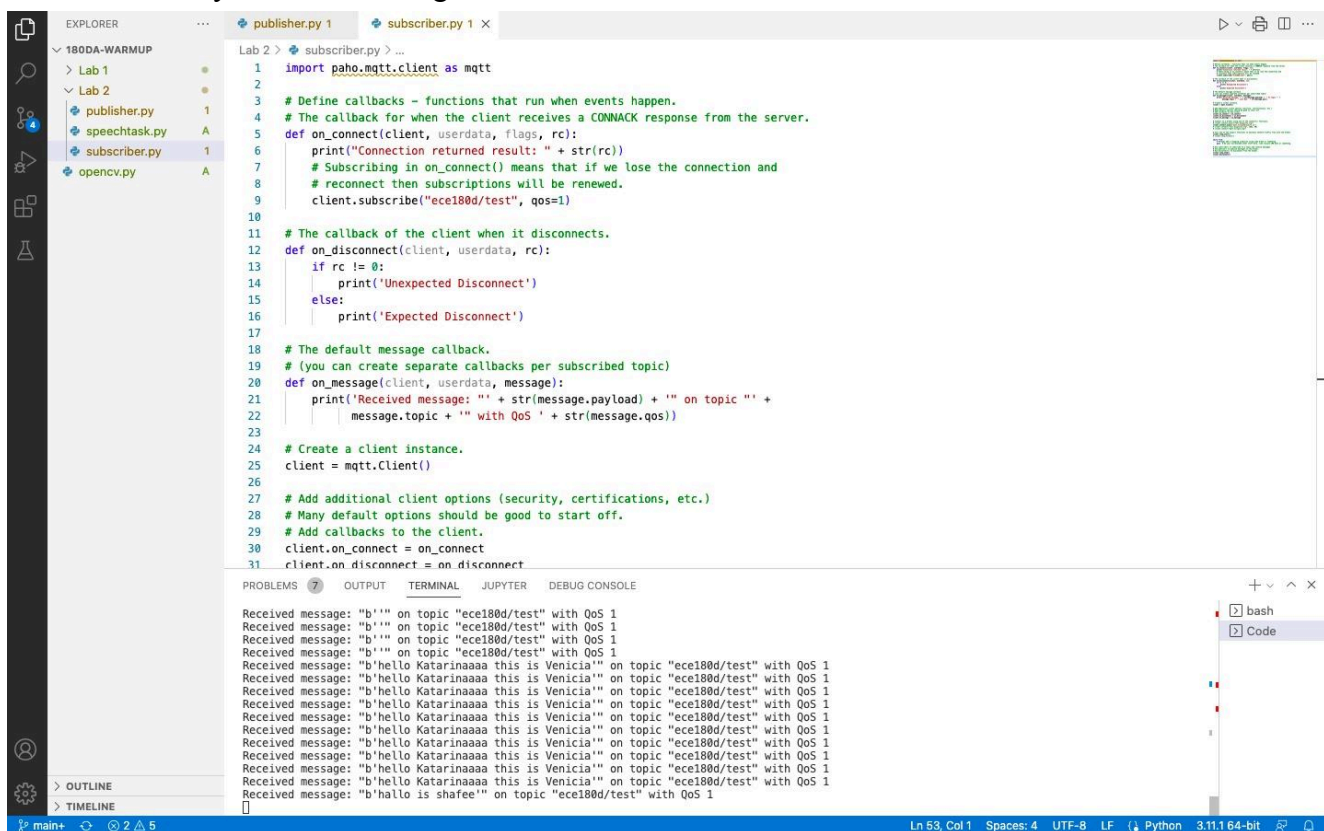


Lab 2 Task

1. Program a working MQTT subscriber and publisher to have two-way communications. If you have something relevant to your project, do it. If not, an easy idea is to ping pong between you and a partner, a counter that increments each time you receive a message (with a delay). Screenshot the terminal and don't forget to mention your partner. In your group of 4, try doing a simple 4-way communications task. Notice the increase in complexity with each additional member. Also note any possible communications lag that you may be having. Consider how to use MQTT for the project. Based on your experiences, what is made possible using MQTT? What seems fairly difficult using MQTT? If you were to use MQTT, what would a reasonable communications lag time be? Would you prefer to use a different method of transmitting data?

Below is a screenshot of my partner, Venicia Massey, sending a message to me, after I sent her a message using MQTT subscriber and publisher. In this screenshot, my partner did not change the range, so it sent her message ten times. However, once we did the communication between the 4 of us in our group, we changed all of our ranges to 1 so that it would only send our messages one time.



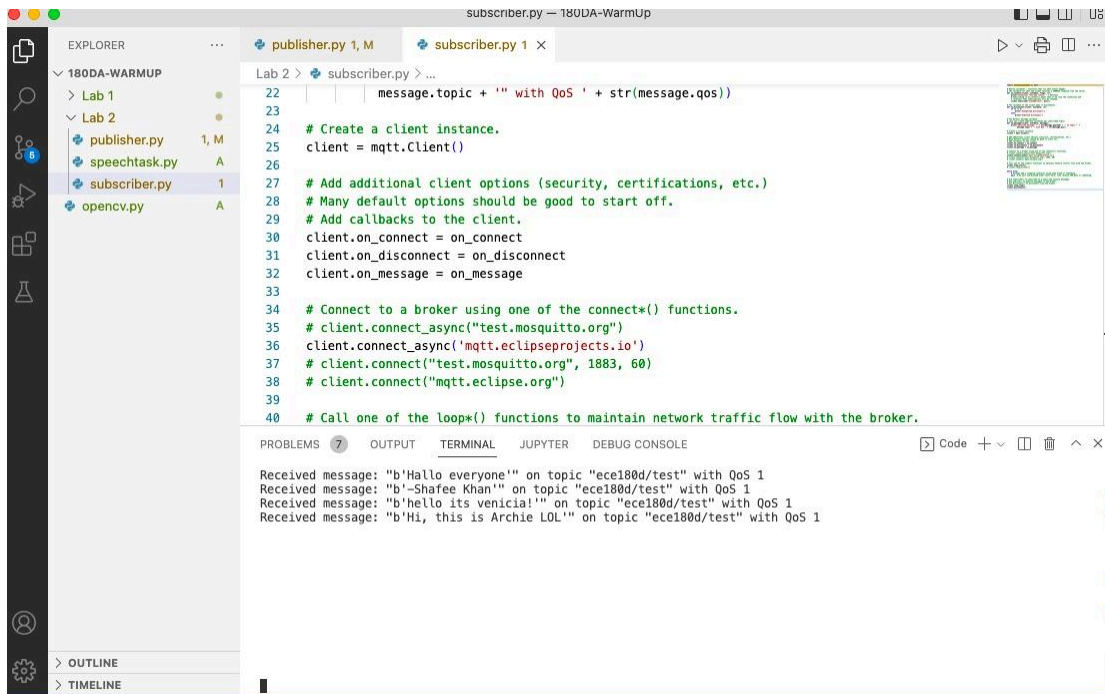
The screenshot shows a VS Code editor with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The file explorer shows a project named '180DA-WARMUP' with subfolders 'Lab 1' and 'Lab 2'. In 'Lab 2', there are files 'publisher.py', 'speechtask.py', 'subscriber.py', and 'opencv.py'. The code editor is open to 'subscriber.py', which contains the following Python code:

```
1 import paho.mqtt.client as mqtt
2
3 # Define callbacks - functions that run when events happen.
4 # The callback for when the client receives a CONNACK response from the server.
5 def on_connect(client, userdata, flags, rc):
6     print("Connection returned result: " + str(rc))
7     # Subscribing in on_connect() means that if we lose the connection and
8     # reconnect then subscriptions will be renewed.
9     client.subscribe("ece180d/test", qos=1)
10
11 # The callback of the client when it disconnects.
12 def on_disconnect(client, userdata, rc):
13     if rc != 0:
14         print('Unexpected Disconnect')
15     else:
16         print('Expected Disconnect')
17
18 # The default message callback.
19 # (you can create separate callbacks per subscribed topic)
20 def on_message(client, userdata, message):
21     print('Received message: "' + str(message.payload) + '" on topic "' +
22         message.topic + '" with QoS ' + str(message.qos))
23
24 # Create a client instance.
25 client = mqtt.Client()
26
27 # Add additional client options (security, certifications, etc.)
28 # Many default options should be good to start off.
29 # Add callbacks to the client.
30 client.on_connect = on_connect
31 client.on_disconnect = on_disconnect
```

The terminal at the bottom shows the output of the script, displaying received messages from the topic 'ece180d/test' with QoS 1. The messages are:

```
Received message: "b''" on topic "ece180d/test" with QoS 1
Received message: "b''" on topic "ece180d/test" with QoS 1
Received message: "b''" on topic "ece180d/test" with QoS 1
Received message: "b''" on topic "ece180d/test" with QoS 1
Received message: "b'hello Katarinaaaa this is Venicia'" on topic "ece180d/test" with QoS 1
Received message: "b'hello Katarinaaaa this is Venicia'" on topic "ece180d/test" with QoS 1
Received message: "b'hello Katarinaaaa this is Venicia'" on topic "ece180d/test" with QoS 1
Received message: "b'hello Katarinaaaa this is Venicia'" on topic "ece180d/test" with QoS 1
Received message: "b'hello Katarinaaaa this is Venicia'" on topic "ece180d/test" with QoS 1
Received message: "b'hello Katarinaaaa this is Venicia'" on topic "ece180d/test" with QoS 1
Received message: "b'hello Katarinaaaa this is Venicia'" on topic "ece180d/test" with QoS 1
Received message: "b'hello Katarinaaaa this is Venicia'" on topic "ece180d/test" with QoS 1
Received message: "b'hello Katarinaaaa this is Venicia'" on topic "ece180d/test" with QoS 1
Received message: "b'hello Katarinaaaa this is Venicia'" on topic "ece180d/test" with QoS 1
Received message: "b'hello Katarinaaaa this is Venicia'" on topic "ece180d/test" with QoS 1
Received message: "b'hallo is shafee'" on topic "ece180d/test" with QoS 1
```

Below is a screenshot of me and my 3 groupmates communicating, sending each other messages using MQTT publisher and subscriber. We did a simple 4-way communication by just sending each other greetings.



The screenshot shows a VS Code editor window with a file explorer on the left and a code editor on the right. The file explorer shows a project named '180DA-WARMUP' with a subdirectory 'Lab 2' containing files 'publisher.py', 'speechtask.py', 'subscriber.py', and 'opencv.py'. The code editor shows the 'subscriber.py' file with the following code:

```
22 | message.topic + ' ' with QoS ' + str(message.qos))
23 |
24 | # Create a client instance.
25 | client = mqtt.Client()
26 |
27 | # Add additional client options (security, certifications, etc.)
28 | # Many default options should be good to start off.
29 | # Add callbacks to the client.
30 | client.on_connect = on_connect
31 | client.on_disconnect = on_disconnect
32 | client.on_message = on_message
33 |
34 | # Connect to a broker using one of the connect*() functions.
35 | # client.connect_async("test.mosquitto.org")
36 | client.connect_async('mqtt.eclipseprojects.io')
37 | # client.connect("test.mosquitto.org", 1883, 60)
38 | # client.connect("mqtt.eclipse.org")
39 |
40 | # Call one of the loop*() functions to maintain network traffic flow with the broker.
```

The terminal output at the bottom shows the following messages:

```
Received message: "b'Hallo everyone'" on topic "ece180d/test" with QoS 1
Received message: "b'-Shafee Khan'" on topic "ece180d/test" with QoS 1
Received message: "b'hello its venicia!'" on topic "ece180d/test" with QoS 1
Received message: "b'Hi, this is Archie LOL'" on topic "ece180d/test" with QoS 1
```

MQTT makes it possible to communicate in a structured, simple way. The publisher is the one who has information to share, and the subscriber is the one who receives that specific information. The messages are organized into topics, making the communication organized. The subscribers can only subscribe to the topics they are interested in. Through MQTT, messages are sent in a lightweight format, making it an easy way to communicate through various devices.

On the other hand, MQTT is not always persistent. For example, there were a couple times I was the publisher and published a message when the subscribers were not subscribed. This led to my message getting lost. Another aspect to take into account, is if MQTT is able to manage a large number of devices. For our project, if we have various devices subscribed, it might cause issues or delays. Also, the publisher and subscriber communication only worked when connected to UCLA Wifi. At first, I was on the UCLA Web, and the subscribers were not receiving my message.

A reasonable lag time for MQTT could be the message size. Larger messages take a longer time to transmit over the network. Also, the location of where my group is working can lead to lagging. Having the publisher closer to the subscribers reduces the lag time.

Although MQTT is not always persistent, I still believe it is an efficient method of transmitting data. My group did not have too much of a difficult time communicating with one another. It just took some time to fully understand how MQTT works.

2. Below is a screenshot of me doing the “Guess the Word” game.

```
print("Sorry, you lose!\nI was thinking of 'apple'")
break
```

[1] ✓ 34.9s Python

... I'm thinking of one of these words:
apple, banana, grape, orange, mango, lemon
You have 3 tries to guess which one.

Guess 1. Speak!
I didn't catch that. What did you say?

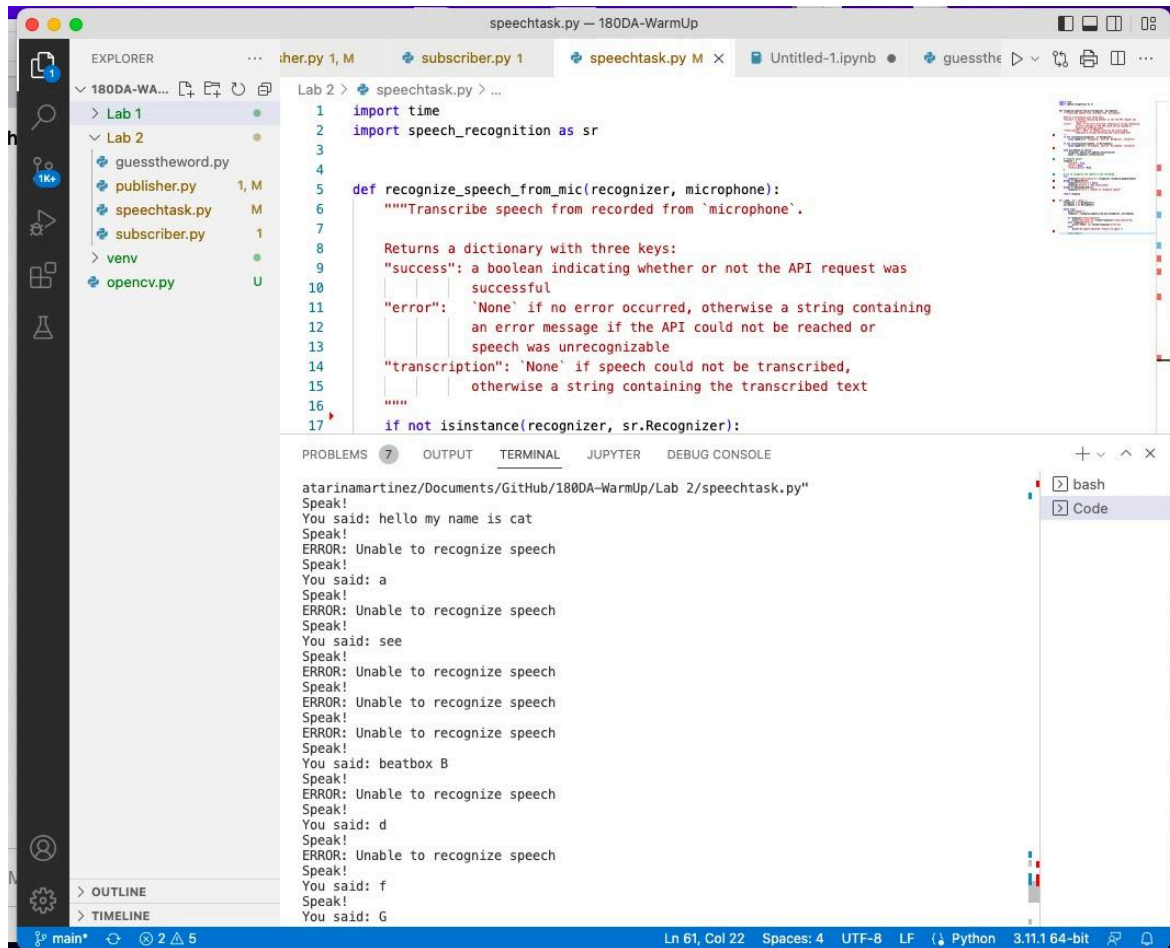
Guess 1. Speak!
You said: mango
Incorrect. Try again.

Guess 2. Speak!
You said: banana
Incorrect. Try again.

Guess 3. Speak!
You said: Orange
Sorry, you lose!
I was thinking of 'apple'.

I did this game to test my speech program, and it worked fairly well. I moved onto saying phrases and letters. The phrases were picked up quite well. The letters sometimes were not translated correctly. For example, when I said the letter “C”, it typed out “see”. Below is a

screenshot of me testing a phrase and letters.



The screenshot shows a VS Code editor window titled "speechtask.py — 180DA-WarmUp". The Explorer panel on the left shows a file structure with "Lab 1" and "Lab 2" folders, and files like "guessstheword.py", "publisher.py", "speechtask.py", "subscriber.py", "venv", and "opencv.py". The main editor area shows the code for "speechtask.py":

```
1 import time
2 import speech_recognition as sr
3
4
5 def recognize_speech_from_mic(recognizer, microphone):
6     """Transcribe speech from recorded from `microphone`.
7
8     Returns a dictionary with three keys:
9     "success": a boolean indicating whether or not the API request was
10     successful
11     "error": 'None' if no error occurred, otherwise a string containing
12     an error message if the API could not be reached or
13     speech was unrecognizable
14     "transcription": 'None' if speech could not be transcribed,
15     otherwise a string containing the transcribed text
16
17     """
18     if not isinstance(recognizer, sr.Recognizer):
```

The TERMINAL panel at the bottom shows the output of the program:

```
atarinamartinez/Documents/GitHub/180DA-WarmUp/Lab 2/speechtask.py"
Speak!
You said: hello my name is cat
Speak!
ERROR: Unable to recognize speech
Speak!
You said: a
Speak!
ERROR: Unable to recognize speech
Speak!
You said: see
Speak!
ERROR: Unable to recognize speech
Speak!
ERROR: Unable to recognize speech
Speak!
ERROR: Unable to recognize speech
Speak!
You said: beatbox B
Speak!
ERROR: Unable to recognize speech
Speak!
You said: d
Speak!
ERROR: Unable to recognize speech
Speak!
You said: f
Speak!
You said: G
```

The performance started taking a hit when I started saying letters instead of words and phrases; however, some of the letters worked. A long phrase can work. I practiced some long sentences, and it did not have that hard of a time translating it to text. I also played a song, not too loud, and it was still able to pick up the words I was saying. I then played a song at full volume which caused the program to not recognize the speech. Ways to improve the performance, is to make sure there is not too much background noise that interferes with what I am saying.

- My group and I can use this speech program in our project if we need one of the players to say a specific phrase. We haven't completely decided on our game yet, but I think the speech program will be very useful.
- I do not think we need the speech recognition to be very complex. As long as it can pick up basic phrases throughout our game, I think we should be good. I believe it would be much more difficult if we expect the program to pick up very complex phrases or sounds.
- For our game, we are going to need accurate recognition. If the program continues to miss recognition throughout the game, it could definitely mess with the game itself and cause setbacks.

- d) I think we will need reasonable confidence that it works well enough, especially before we have other groups play our game. We want to make sure the speech recognition is pretty accurate and consistent, so it does not affect the game progress.