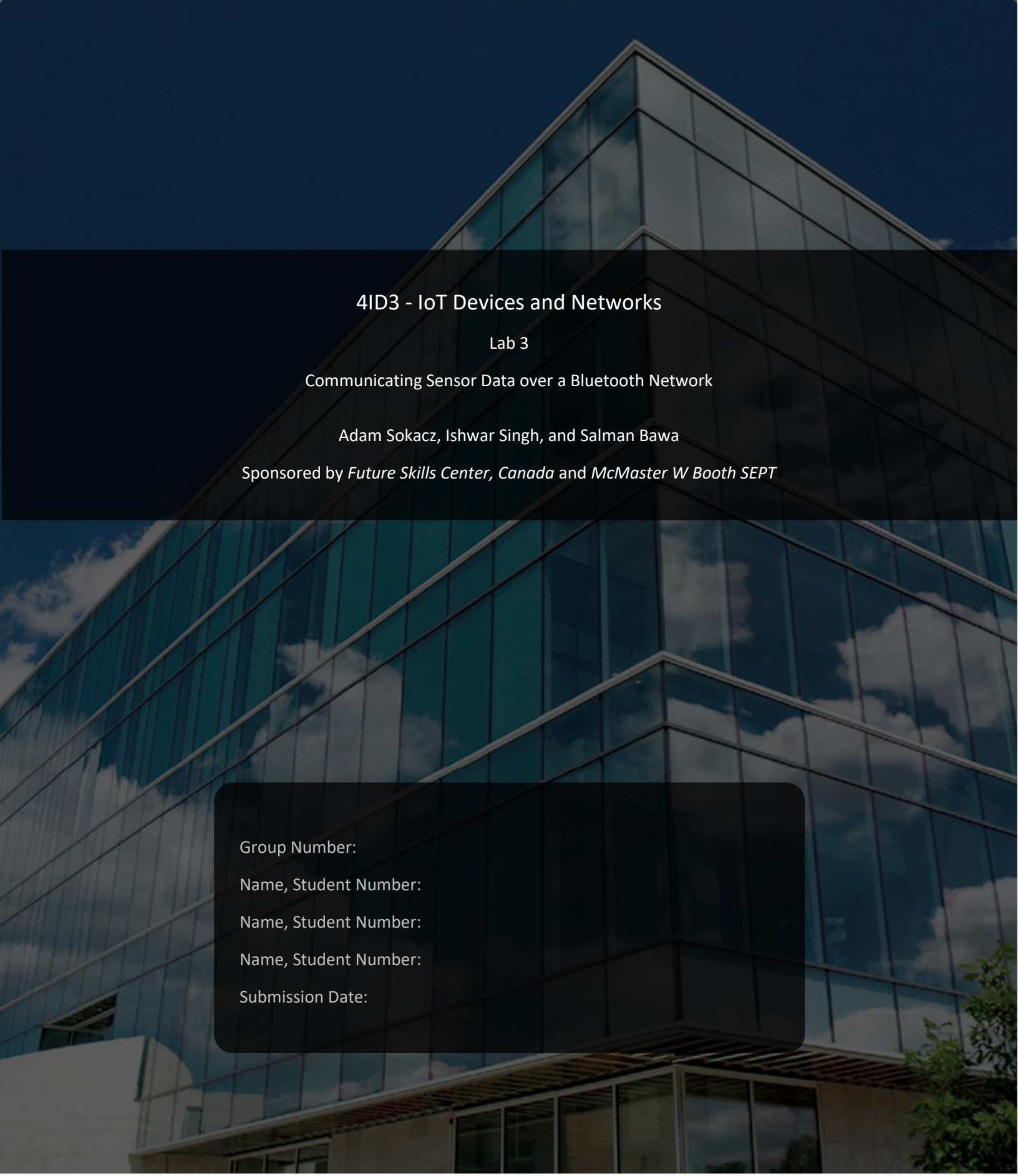


4ID3 - IoT Devices and Networks

Lab 3

Communicating Sensor Data over a Bluetooth Network

Adam Sokacz, Ishwar Singh, and Salman Bawa

Sponsored by *Future Skills Center, Canada* and *McMaster W Booth SEPT*

Group Number:

Name, Student Number:

Name, Student Number:

Name, Student Number:

Submission Date:

W BOOTH SCHOOL OF ENGINEERING PRACTICE AND TECHNOLOGY

McMaster-Mohawk Bachelor of Technology Partnership

McMaster University ENGINEERING

mohawk

Objective

In this lab, sensor data will be read, encoded, and transmitted to a nearby PC using the IoT access technology Bluetooth Classic. This data will then be parsed and routed to both a public MQTT server for use in NodeRED, and a locally run MongoDB database for data storage and analysis.

Contents

Objective 2
Feedback 3
Additional Resources 3
Pre-Lab Questions 4
Post-Lab Questions 4
Exercise A Results: 5
Optional: Exercise B Results: 5
Setting up the Workspace 6
Reading Sensor Data 13
Transmitting Over Bluetooth 24
Viewing a Bluetooth Serial Device 30
Exercise A 35
Parsing Data using Python 36
Installing Python 36
Installing VSCode 37
MQTT Routing 41
Verify MQTT 47
Exercise B 48
Exiting the Python Application 48
Optional: Connecting to a Database 50
Exercise C 58
Pushing Changes to GitHub 59

Feedback

Q1 - What would you rate the difficulty of this lab?

(1 = easy, 5 = difficult)

1 **2** **3** **4** **5**

Comments about the difficulty of the lab:

Q2 - Did you have enough time to complete the lab within the designated lab time?

YES **NO**

Q3 - How easy were the lab instructions to understand?

(1 = easy, 5 = unclear)

1 **2** **3** **4** **5**

List any unclear steps:

Q4 - Could you see yourself using the skills learned in this lab to tackle future engineering challenges?

(1 = no, 5 = yes)

1 **2** **3** **4** **5**

Additional Resources

Lab GitHub Repo (https://github.com/sokacza/4ID3)

Arduino Programming Refresher (https://youtu.be/CbJHL_P5RJ8)

Mosquitto MQTT Broker Tutorial (https://youtu.be/DH-VSAACtBk)

NodeRED Fundamentals Tutorial (https://youtu.be/3AR432bguOY)

ESP32 Overview (https://youtu.be/UuxBfKA3U5M)

Pre-Lab Questions

Q1 - In your own words, describe the master-slave messaging pattern. What role does the master device play? What role do the slave devices play? How many slave devices can be coordinated by a single master device?

Q2 - Compare and contrast Bluetooth Classic vs BLE. What are advantages and drawbacks of each technology?

Q3 - What layer of the OSI model does Bluetooth technology reside in?

Q4 - What is a network gateway? What characteristics must the device have to be considered a gateway device?

Post-Lab Questions

Q1 - What network topology would best describe the IoT network built in this lab? Explain with a diagram.

Q2 - If the microcontroller is transmitting data to the PC, but the Python script is not running, how does this impact what data is accessible through MQTT?

Q3 - What is the theoretical range of Bluetooth Classic? How might this impact its use in IoT networks?

Q4 - What is a relational database? What is a non-relational database? How do they differ? What type of database is MongoDB?

Q5 - List 3 security concerns of this Bluetooth network. How might these concerns be mitigated?

Q6 - Write a brief LinkedIn post about key learning takeaways from this lab.

Exercise A Results:

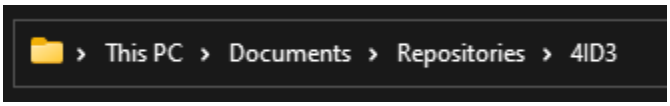
Exercise B Results:

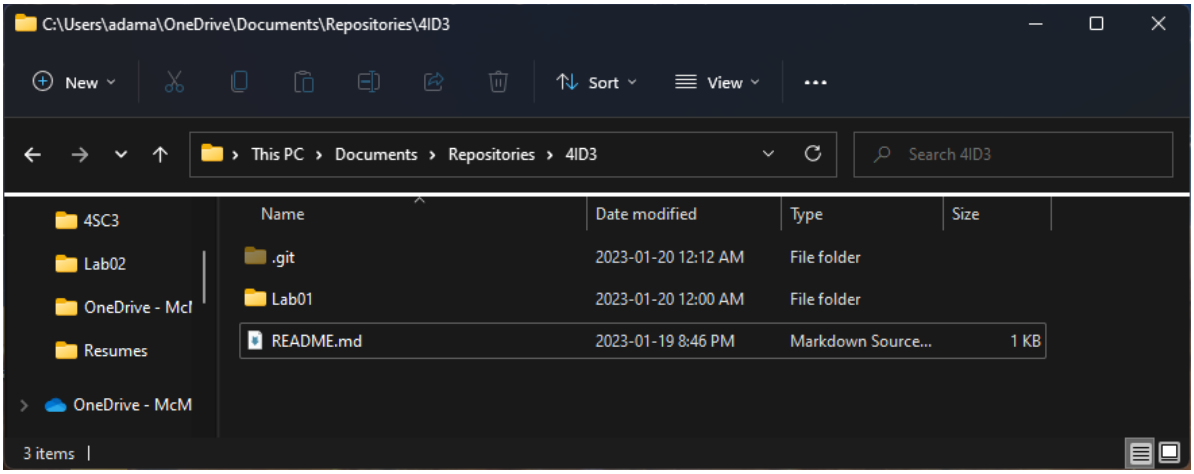
Optional: Exercise C Results:

Setting up the Workspace

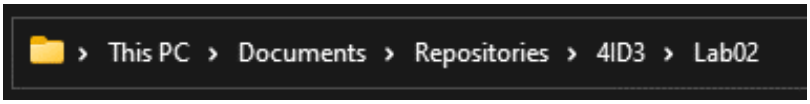
Each lab, we will be creating a new folder in the local git repository that was created in the provided pre-lab to store and document technologies that you have worked on.

Navigate to your local git repository for this course.

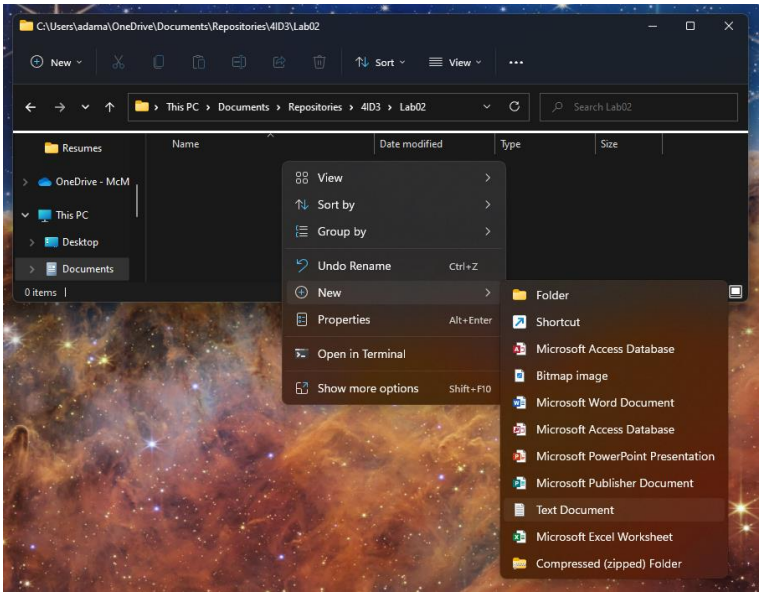




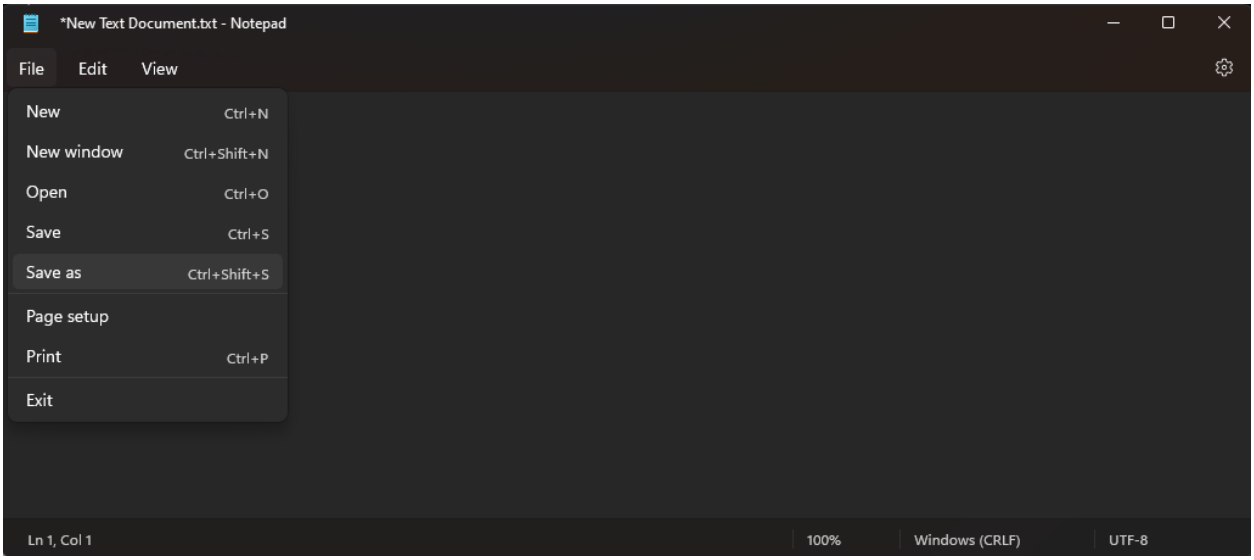
Create a new folder named **Lab02**. Navigate inside this folder.



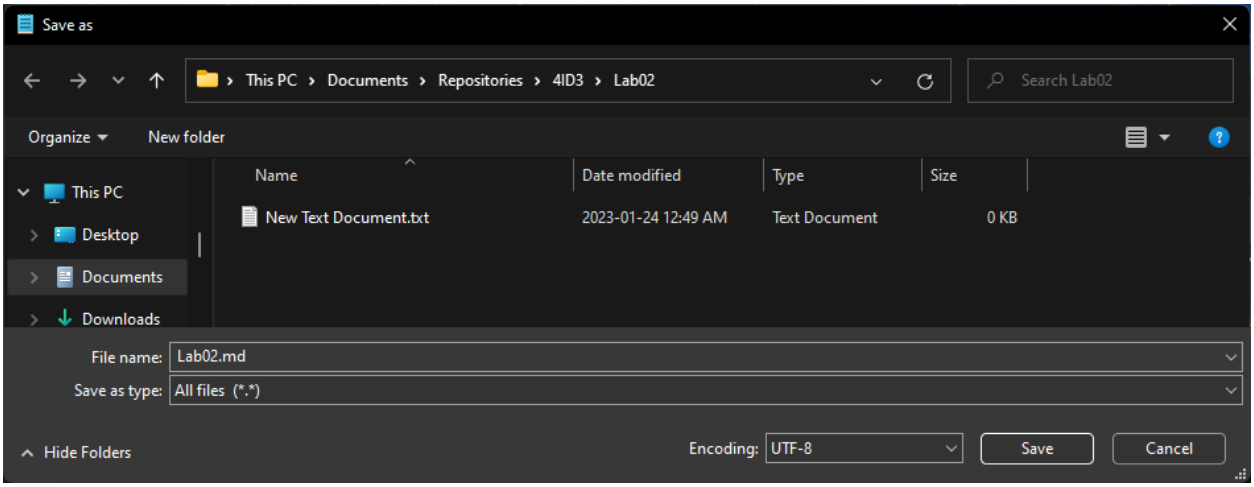
Create a new text file in the folder.



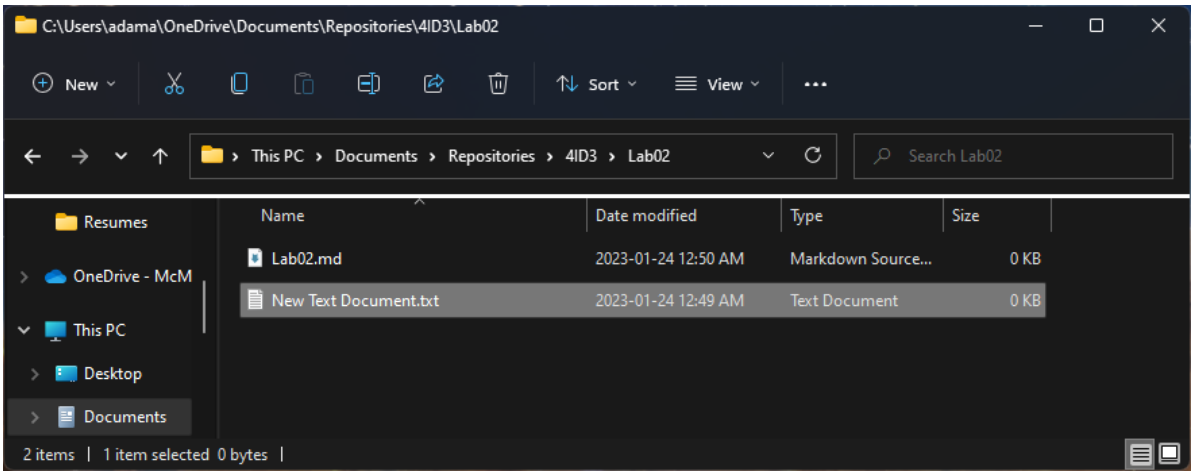
Press **File > Save as.**

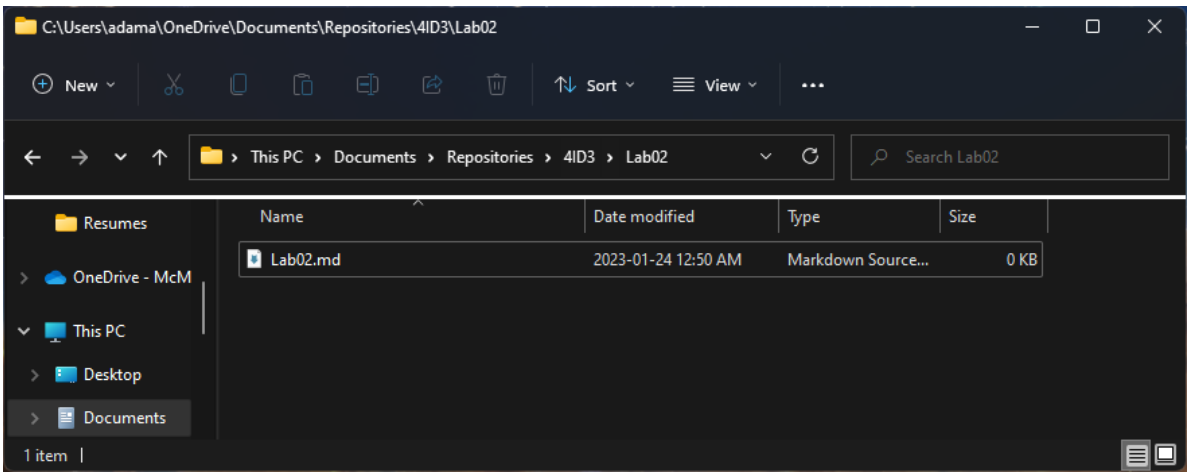


Save it as **Lab02.md**. Ensure that the **Save as type** is set to **All files (*.*)**.

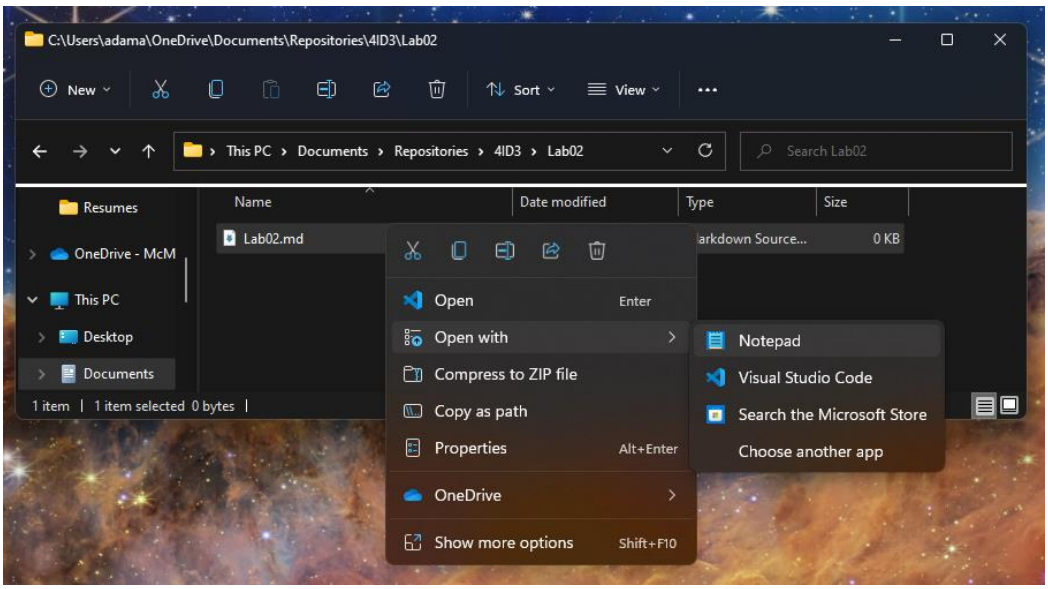


Now, you should have two files, a **text file** and a **markdown file**. Delete the text file.





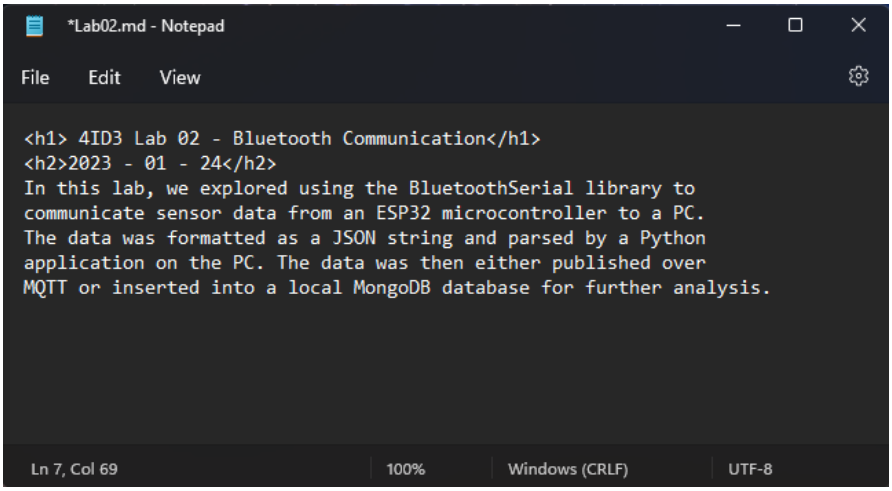
To open the markdown file, **right-click** and select **Open with**. Choose **Notepad**.



Writing markdown documents to explain your code is very similar to HTML. A reference guide can be found here:

https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

Write the following text in the markdown file and save it.



<h1> 4ID3 Lab 02 - Bluetooth Communication</h1>

<h2>2023 - 01 - 24</h2>

In this lab, we explored using the BluetoothSerial library to

communicate sensor data from an ESP32 microcontroller to a PC.

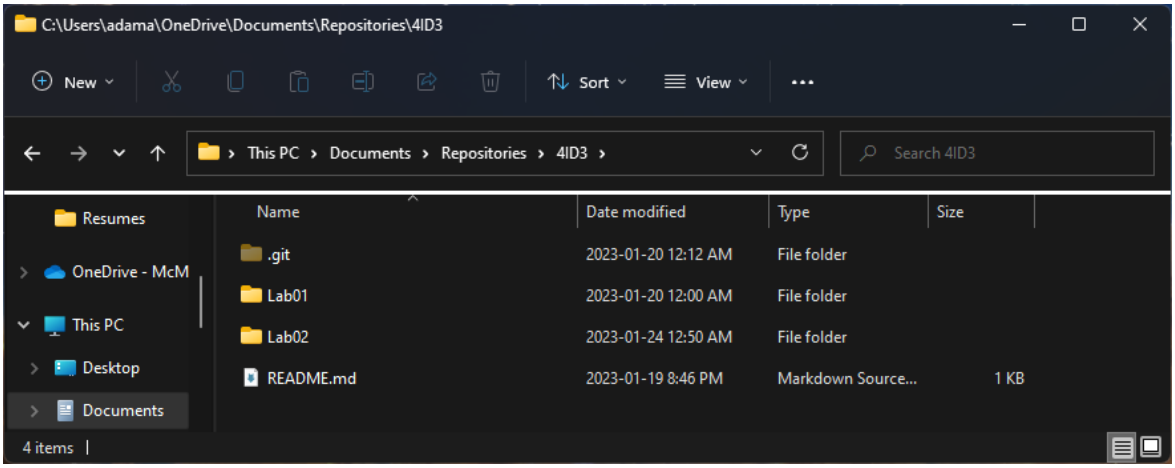
The data was formatted as a JSON string and parsed by a Python

application on the PC. The data was then either published over

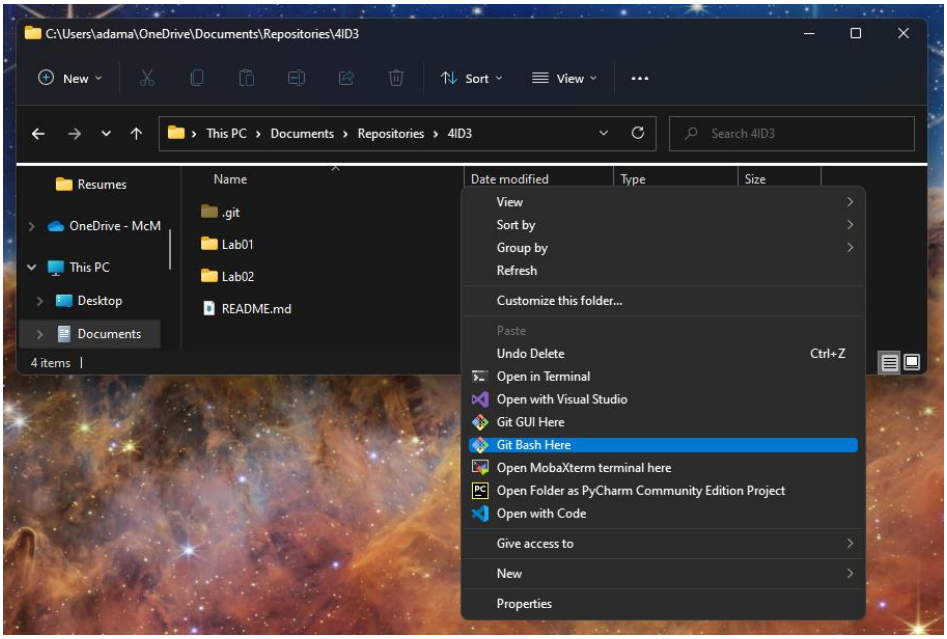
MQTT or inserted into a local MongoDB database for further analysis.

Save the file.

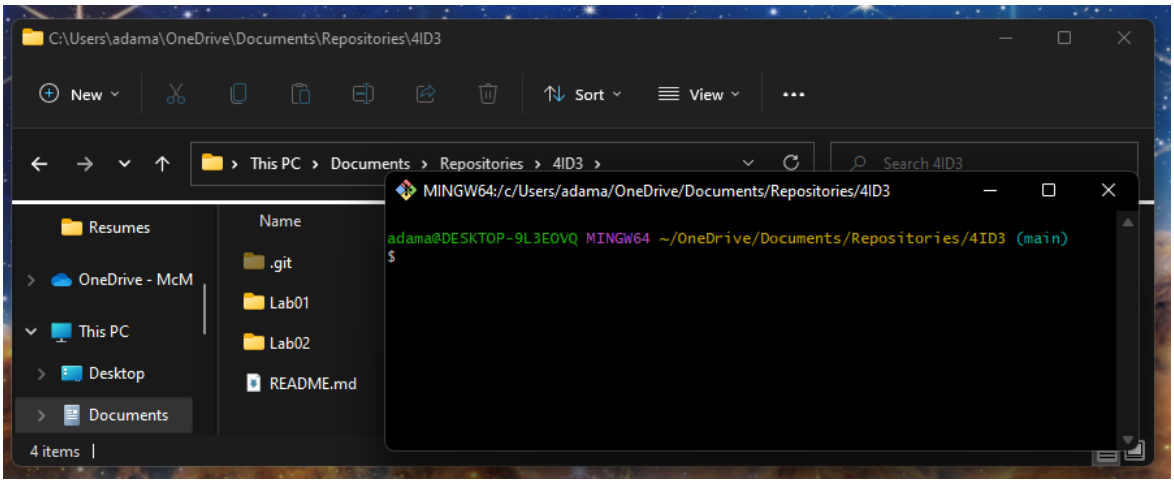
Navigate to the **root folder** (*4ID3/, root main highest-level folder*) of your local repository.



Right-click in the root folder of your local repository and launch **git bash**.



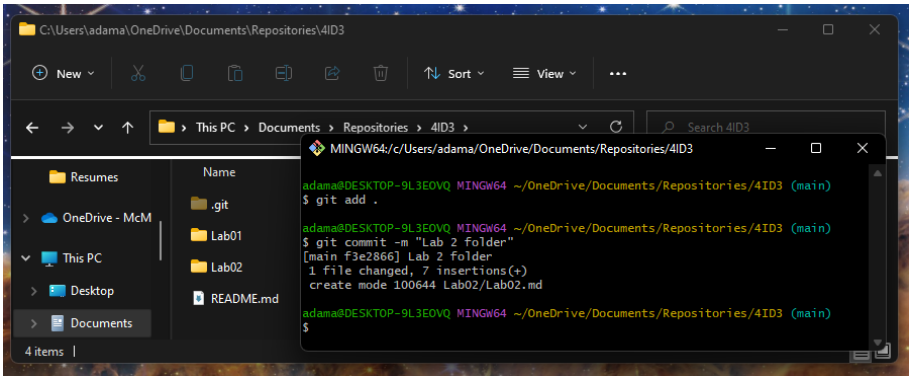
First, we need to add all the changes to the index that will be synced with GitHub. This will be done with the git add command.



git add .

The period '.' Is used as a shorthand for selecting all changes.

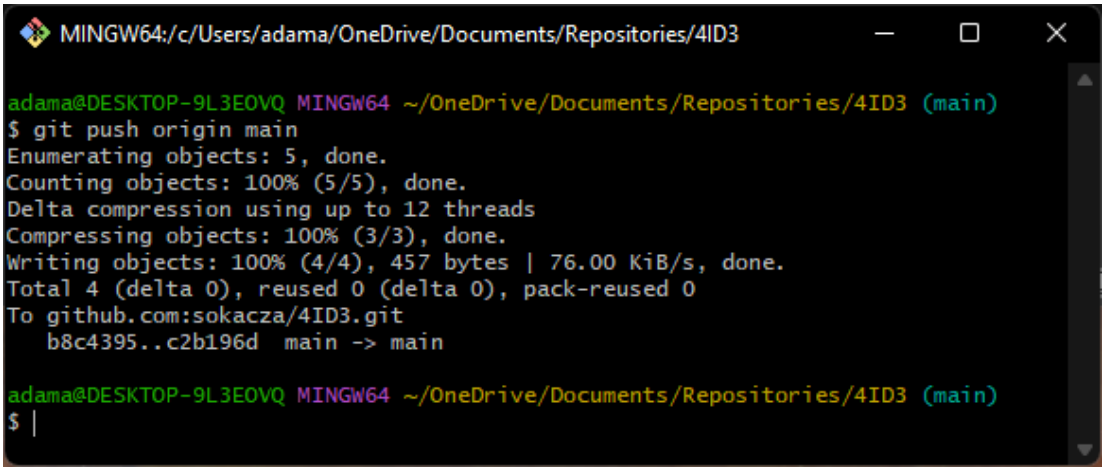
Next, when we are happy with the changes we chose to upload, we can use the commit command to package them to be synced.



git commit -m "Lab 2 folder"

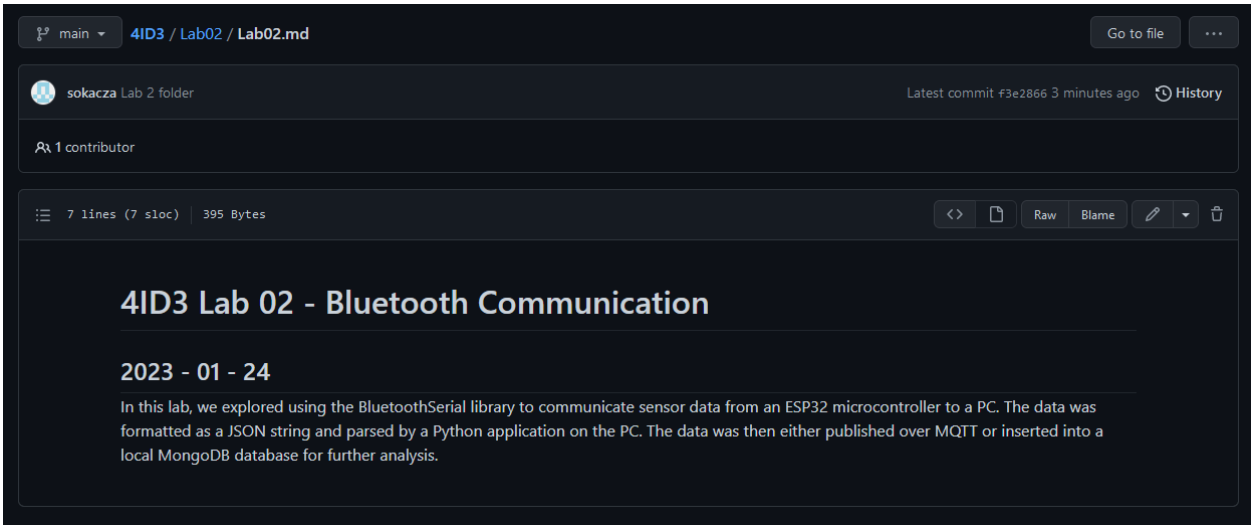
The '-m' flag stands for message, and it adds a message that explains what changes were made.

Lastly, to sync your local git repository with GitHub, use the git push command.

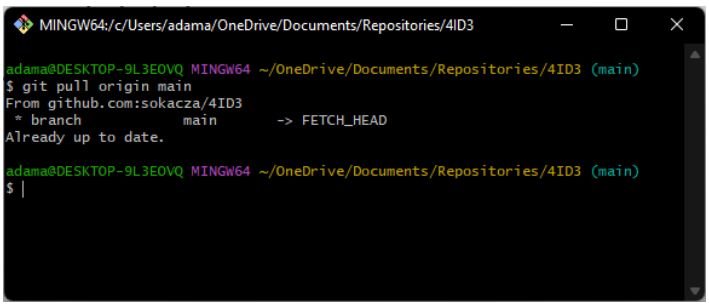


git push origin main

Now, log into GitHub and verify that the changes have been made.



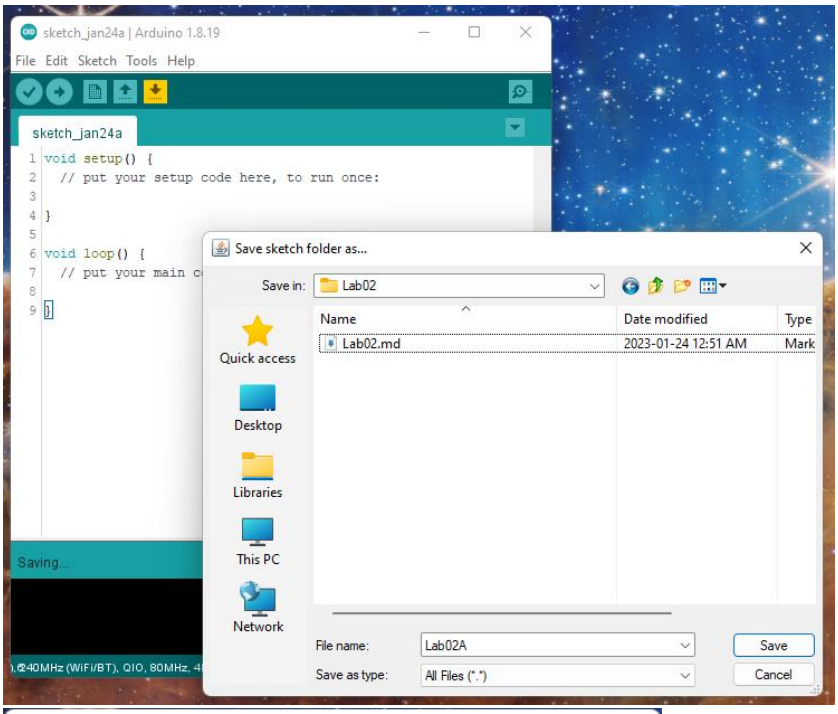
Now, if you are collaborating and wish to sync your local git repo with the remote GitHub repo, use the git pull command. In this case, we see that our local git repo is already up-to-date.



Reading Sensor Data

The goal of this section of the lab is to read data from 2 sensors and print them to the serial monitor, before we communicate to a server.

Launch the Arduino IDE and **Save as** into your lab 2 folder.

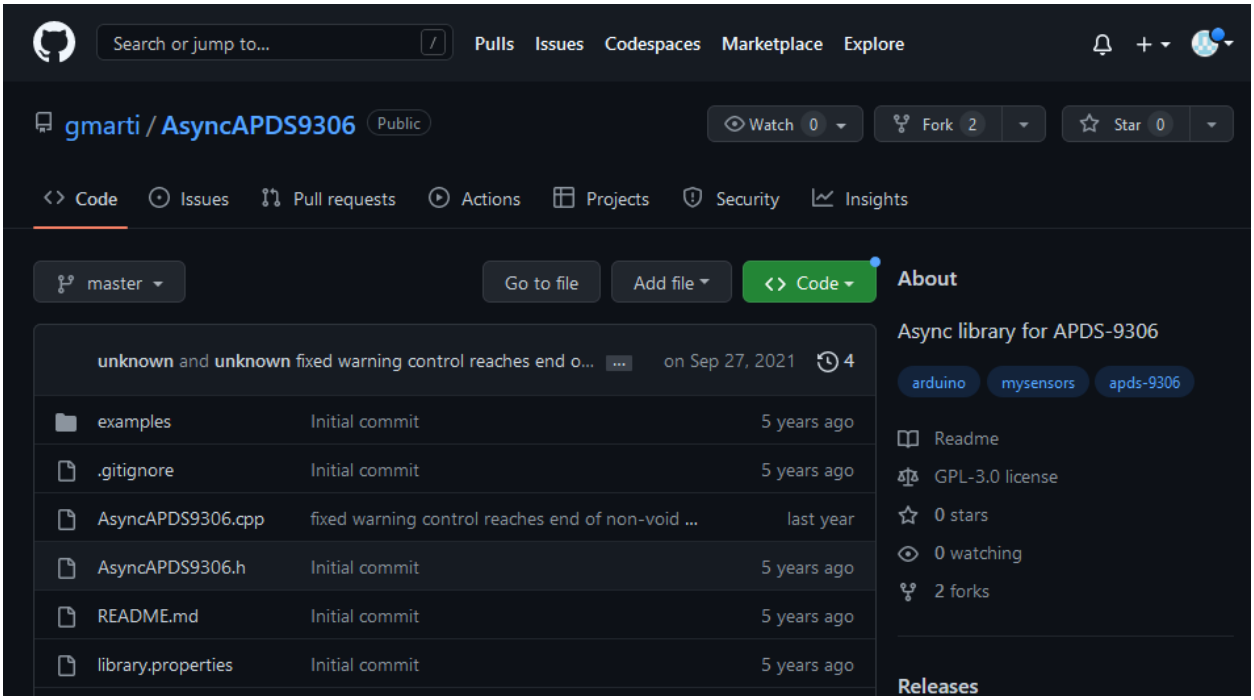


The 2 sensors we will be using are as follows:

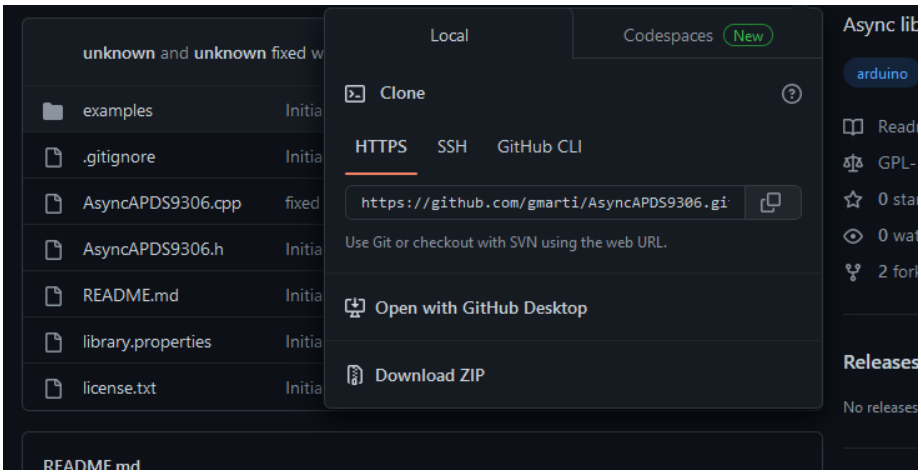
- APDS9306 for Light Intensity
- Si7020-A20 for Temperature

The APDS9306 library must be downloaded from GitHub. Navigate to the following link:

https://github.com/gmarti/AsyncAPDS9306

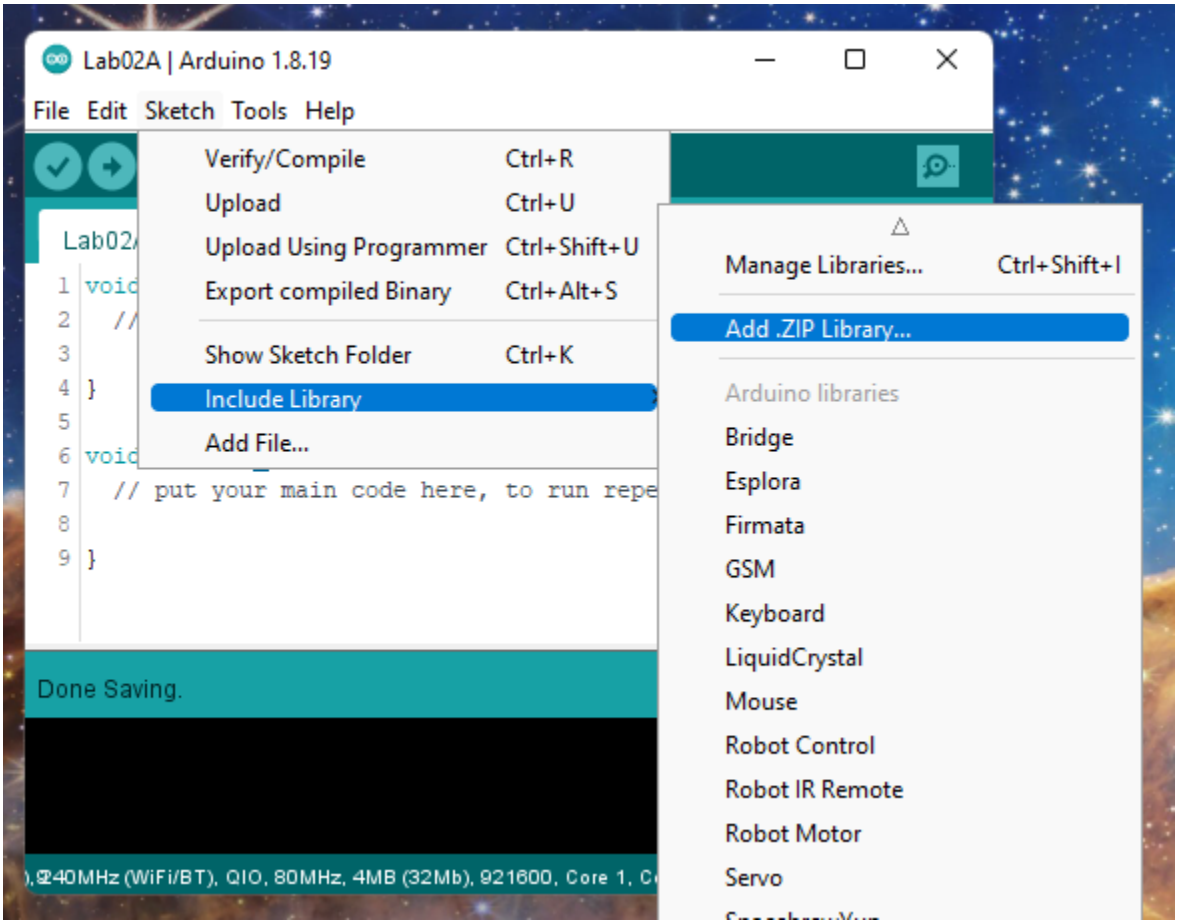


Press **Code** > **Download Zip.**

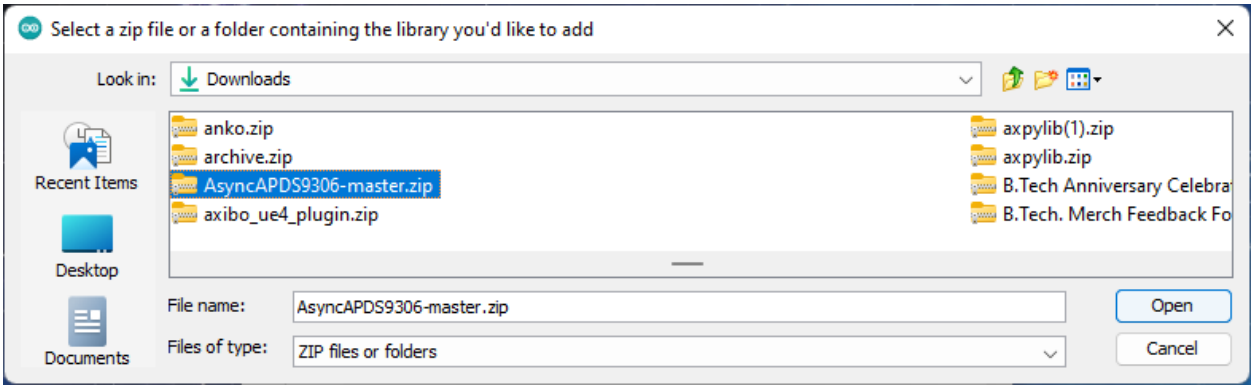


Navigate back to the **Arduino IDE**.

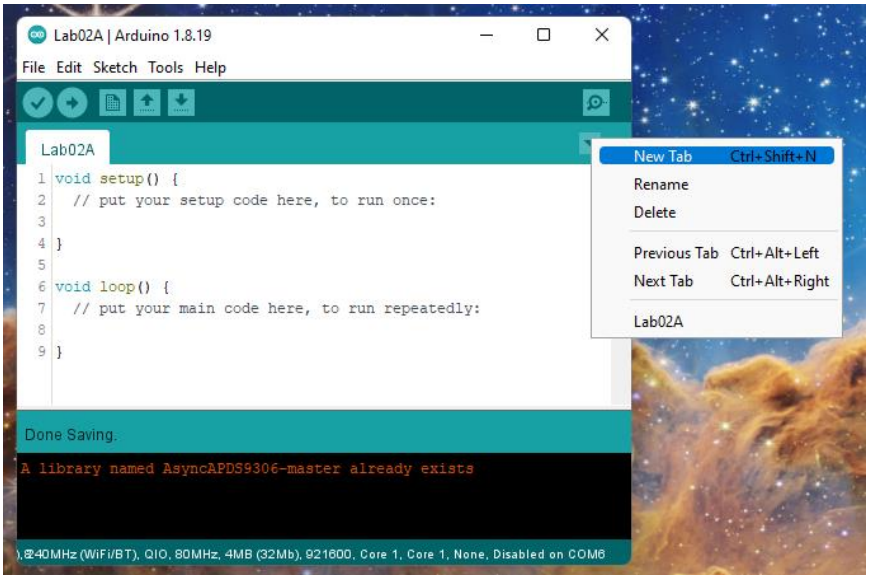
Navigate to **Sketch > Include Library > Add .ZIP Library**.

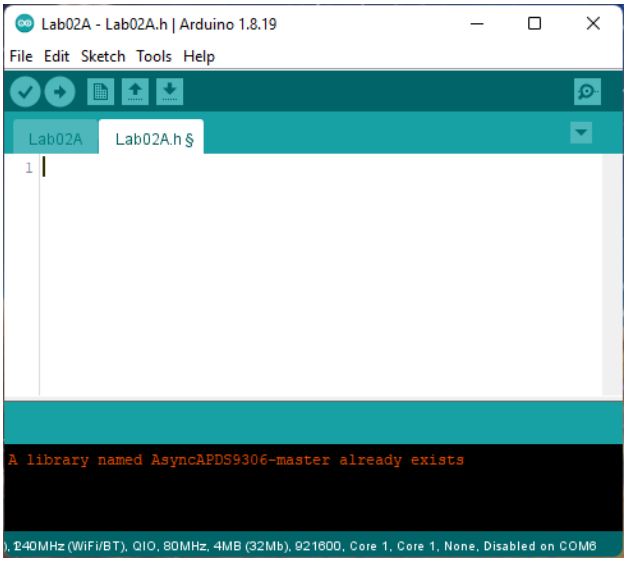


Find and open the downloaded library.



Create a new header file. Name this file **Lab02.h**.





In this file, we will keep our dependencies, preprocessor macros, global variables, and global objects.

Lab02A | Lab02A.h §

```
1  //Libraries
2  #include <Arduino.h>
3  #include <Wire.h>
4  #include <AsyncAPDS9306.h>
5
6  //IIC Addresses for Temperature Sensor
7  #define ADDR (byte)(0x40)
8  #define TMP_CMD (byte)(0xF3)
9
10 //Sample frequency
11 #define DELAY_BETWEEN_SAMPLES_MS 5000
12
13 //Device information
14 String groupName = "GroupA";
15 String deviceName = "DeviceA";
16
17 //Instantiating sensor object and configuration
18 AsyncAPDS9306 lightSensor;
19 const APDS9306_ALS_GAIN_t aGain = APDS9306_ALS_GAIN_1;
20 const APDS9306_ALS_MEAS_RES_t aTime = APDS9306_ALS_MEAS_RES_16BIT_25MS;
21
```

Lastly, lets instantiate some global objects for classes that we will be using throughout the code.

This concludes the header file. Let's move back to the implementation file.

Firstly, include our header file.

Lab02A § | Lab02A.h

```
1  #include "Lab02A.h"
2
3  void setup() {
4    // put your setup code here, to run once:
5
6  }
7
8  void loop() {
9    // put your main code here, to run repeatedly:
10
11 }
12
```

Next, set up the void setup() function.

Lab02A § Lab02A.h §

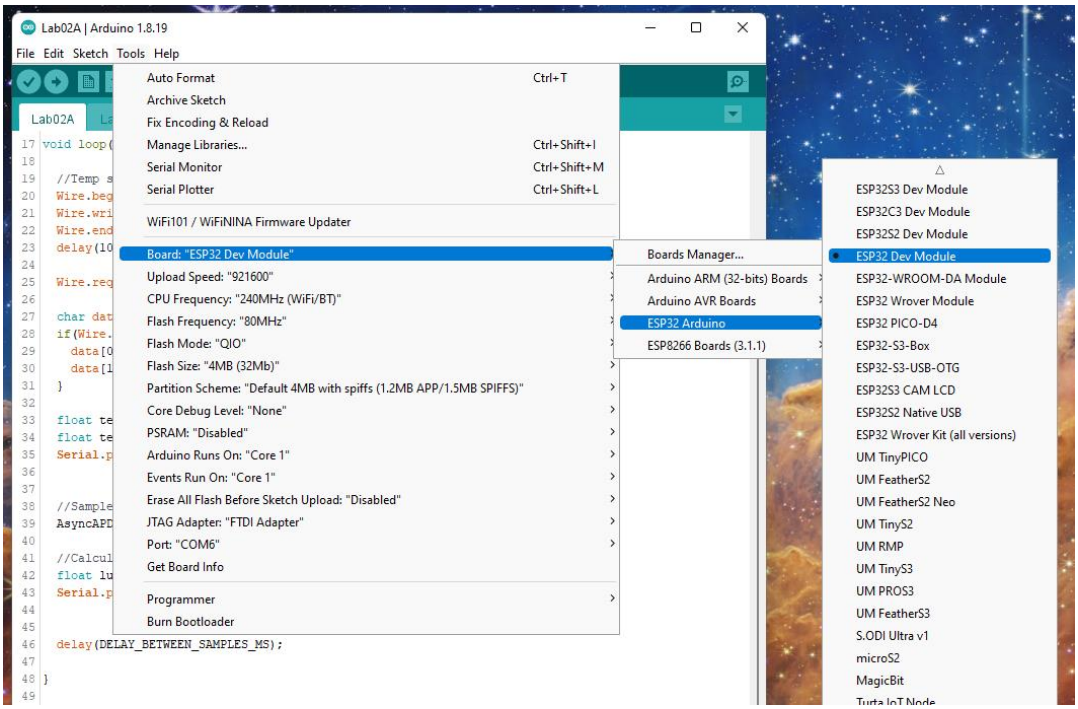
```
#include "Lab02A.h"

void setup() {
  Serial.begin(9600);
  Serial.print("\n\n-------------------------\n"
    + groupName + " : " + deviceName + "\n-------------------------\n\n");

  Wire.begin();
  Wire.beginTransmission(ADDR);
  Wire.endTransmission();
  delay(300);

  lightSensor.begin(aGain, aTime);

}

```

Moving onto the void loop() function, it should be noticed that both the Temperature and Light Intensity sensors are being polled in different ways.

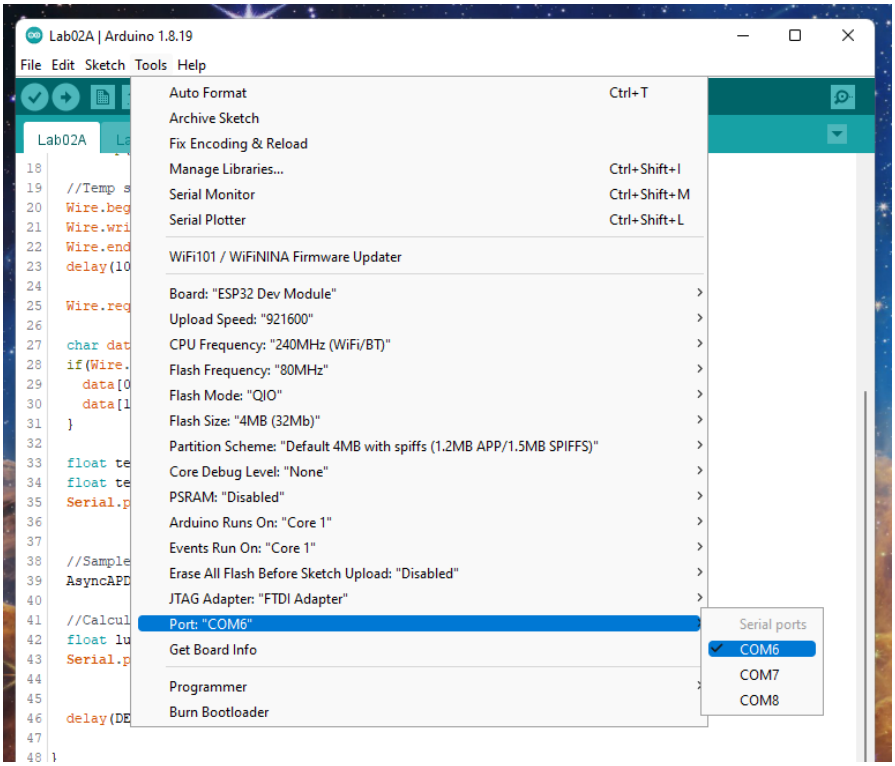
Lab02A | Lab02A.h

```
void loop() {

  //Temp sensor
  Wire.beginTransmission(ADDR);
  Wire.write(TMP_CMD);
  Wire.endTransmission();
  delay(100);

  Wire.requestFrom(ADDR, 2);

  char data[2];
  if(Wire.available() == 2){
    data[0] = Wire.read();
    data[1] = Wire.read();
  }

  float temp = ((data[0] * 256.0) + data[1]);
  float tempC = ((175.72 * temp) / 65536.0) - 46.85;
  Serial.println("Temperature: " + String(tempC) + " degC");


  //Sample light sensor
  AsyncAPDS9306Data lightData = lightSensor.syncLuminosityMeasurement();

  //Calculate luminosity
  float lux = lightData.calculateLux();
  Serial.println("Luminosity: " + String(lux) + " Lux");


  delay(DELAY_BETWEEN_SAMPLES_MS);

}

```

To upload to the board, change the board to ESP32 Dev Module.

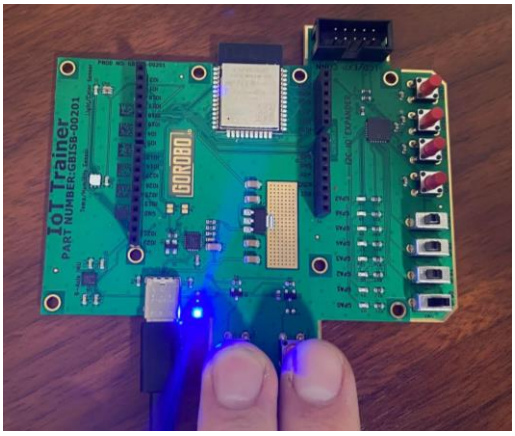


Leave the default communication settings the same, with the exception of the COM port. Select the COM port that your microcontroller is connected to.

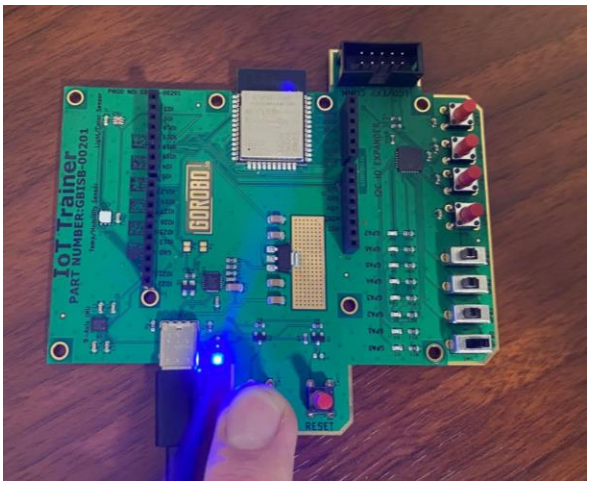


Uploading to the MacIoT board can be somewhat unintuitive. Follow the following steps:

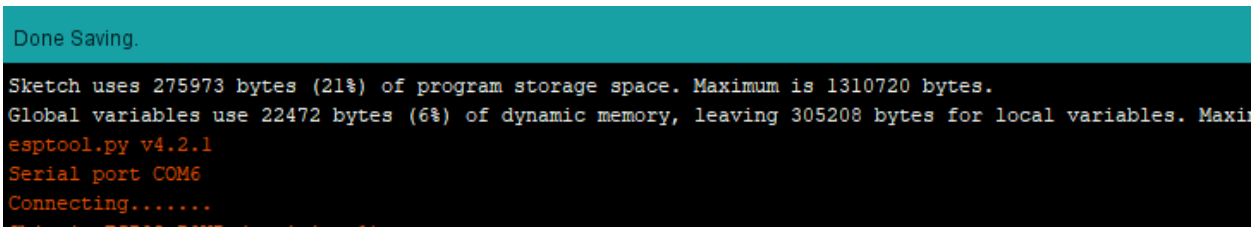
1 - Hold down **both** the **FLASH** and **RESET** buttons

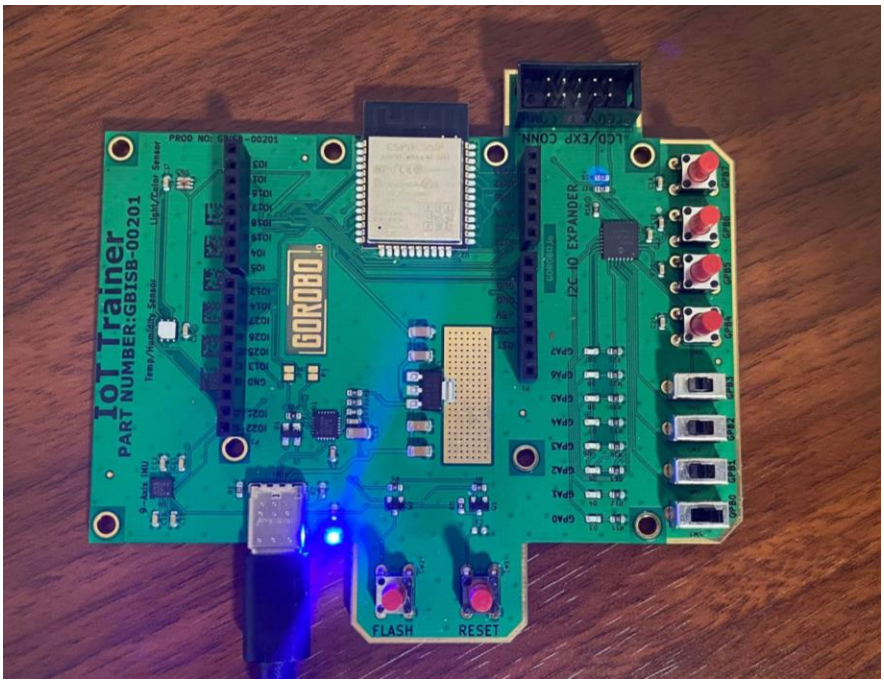


2 - After 2 seconds, **release** the **RESET** button. Continue to hold the **FLASH** button



3 - Once the code finishes compiling and begins to upload, **release** the **FLASH** button.

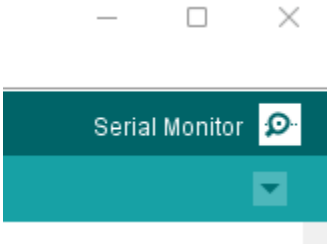


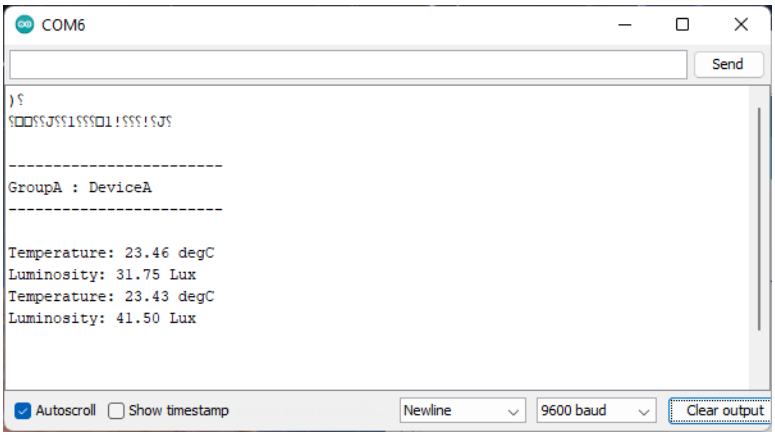


A successful upload should look like this:

```
Done Saving.
Compressed 3072 bytes to 146...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (146 compressed) at 0x00008000 in 0.1 seconds (effective 423.1 kbit/s)...
Hash of data verified.
Compressed 8192 bytes to 47...
Writing at 0x0000e000... (100 %)
Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.1 seconds (effective 587.1 kbit/s)...
Hash of data verified.
Compressed 276368 bytes to 154169...
Writing at 0x00010000... (10 %)
Writing at 0x0001c669... (20 %)
Writing at 0x00024d75... (30 %)
Writing at 0x0002a1a7... (40 %)
Writing at 0x0002f6c4... (50 %)
Writing at 0x00034d01... (60 %)
Writing at 0x0003d2c7... (70 %)
Writing at 0x0004666b... (80 %)
Writing at 0x0004ba9a... (90 %)
Writing at 0x00051371... (100 %)
Wrote 276368 bytes (154169 compressed) at 0x00010000 in 2.6 seconds (effective 854.4 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

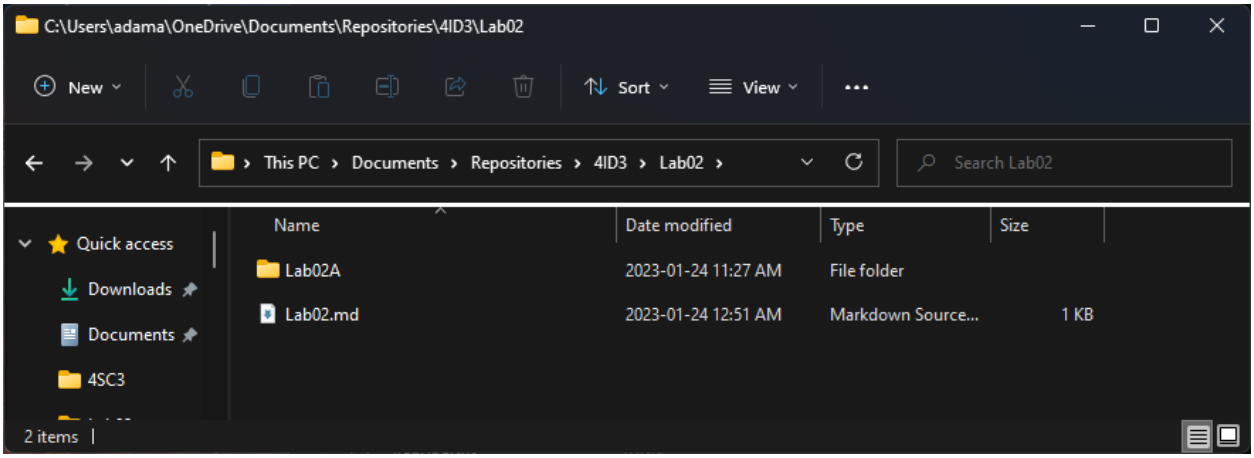
After the upload is complete (NOT DURING), launch the **Serial monitor** to view the microcontroller output.

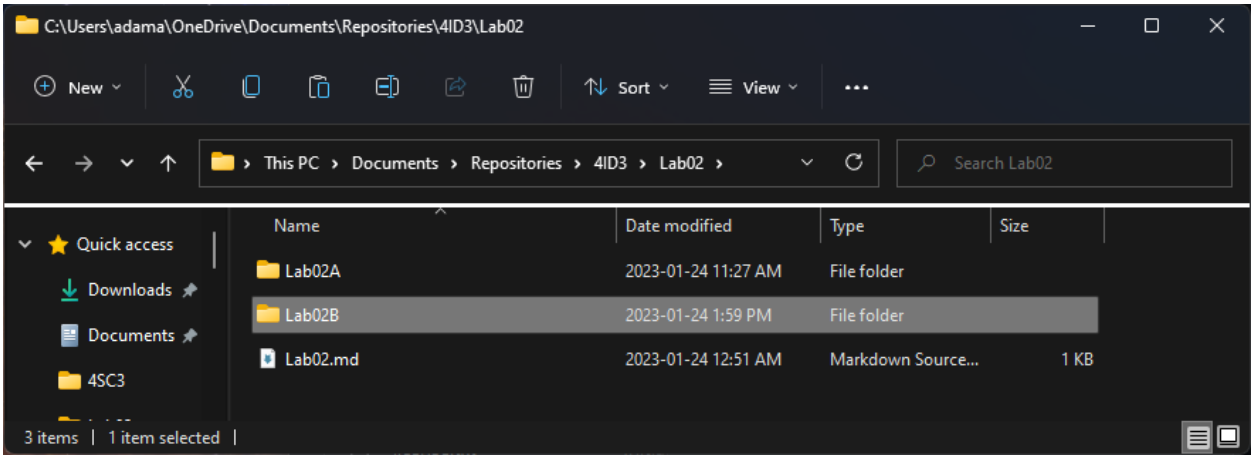




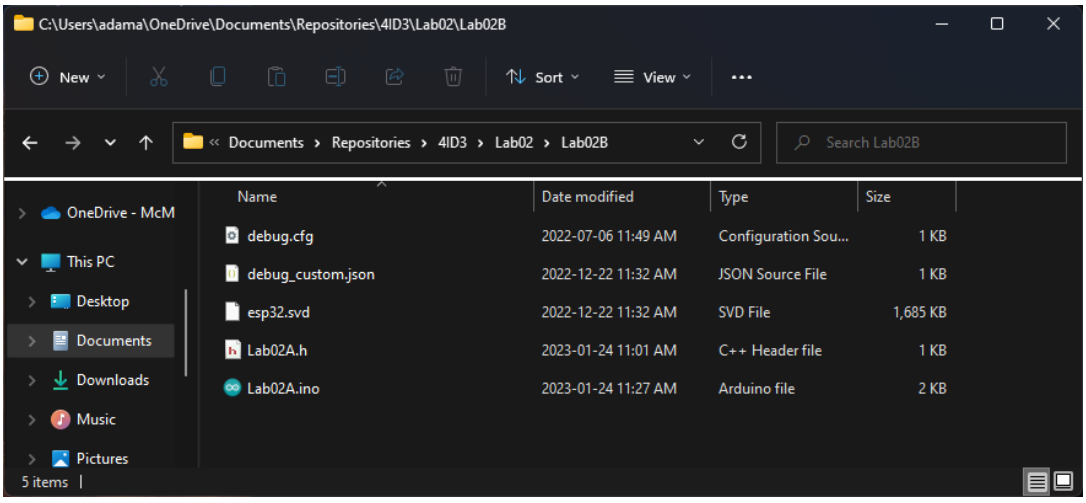
Transmitting Over Bluetooth

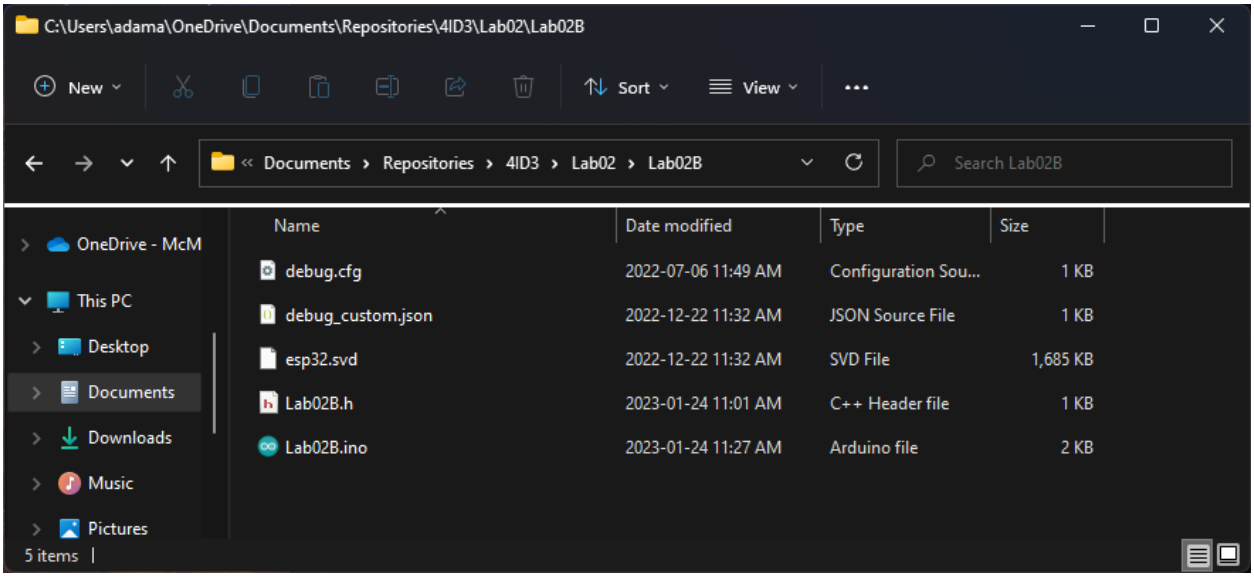
Navigate back to your **Lab02 folder**. **Copy** Lab02A and **rename** it to **Lab02B**.





Rename the constituent files.





Open the project in **Arduino IDE**.

Rename the header file import to match the new name.

Modify the header file to include the built-in **BluetoothSerial** library. Instantiate a serial object for use in the implementation file.

Lab02B | Lab02B.h §

```
1  //Libraries
2  #include <Arduino.h>
3  #include <Wire.h>
4  #include <AsyncAPDS9306.h>
5  #include "BluetoothSerial.h"
6
7  //IIC Addresses for Temperature Sensor
8  #define ADDR (byte)(0x40)
9  #define TMP_CMD (byte)(0xF3)
10
11 //Sample frequency
12 #define DELAY_BETWEEN_SAMPLES_MS 5000
13
14 //Device information
15 String groupName = "GroupA";
16 String deviceName = "DeviceA";
17
18 //Instantiating sensor object and configuration
19 AsyncAPDS9306 lightSensor;
20 const APDS9306_ALS_GAIN_t aGain = APDS9306_ALS_GAIN_1;
21 const APDS9306_ALS_MEAS_RES_t aTime = APDS9306_ALS_MEAS_RES_16BIT_25MS;
22
23 //Instantiating bluetooth serial object
24 BluetoothSerial SerialBT;
25
```

In the **setup()** function of the implementation file, use the **begin()** method of the **SerialBT** object to initialize the Bluetooth link.

Lab02B | Lab02B.h §

```
#include "Lab02B.h"

void setup() {
  Serial.begin(9600);
  Serial.print("\n\n------------------------\n"
    + groupName + " : " + deviceName + "\n------------------------\n\n");

  Wire.begin();
  Wire.beginTransmission(ADDR);
  Wire.endTransmission();
  delay(300);

  lightSensor.begin(aGain, aTime);

  SerialBT.begin(groupName + " : " + deviceName);
  Serial.println("Ready for bluetooth connection!");

}

```

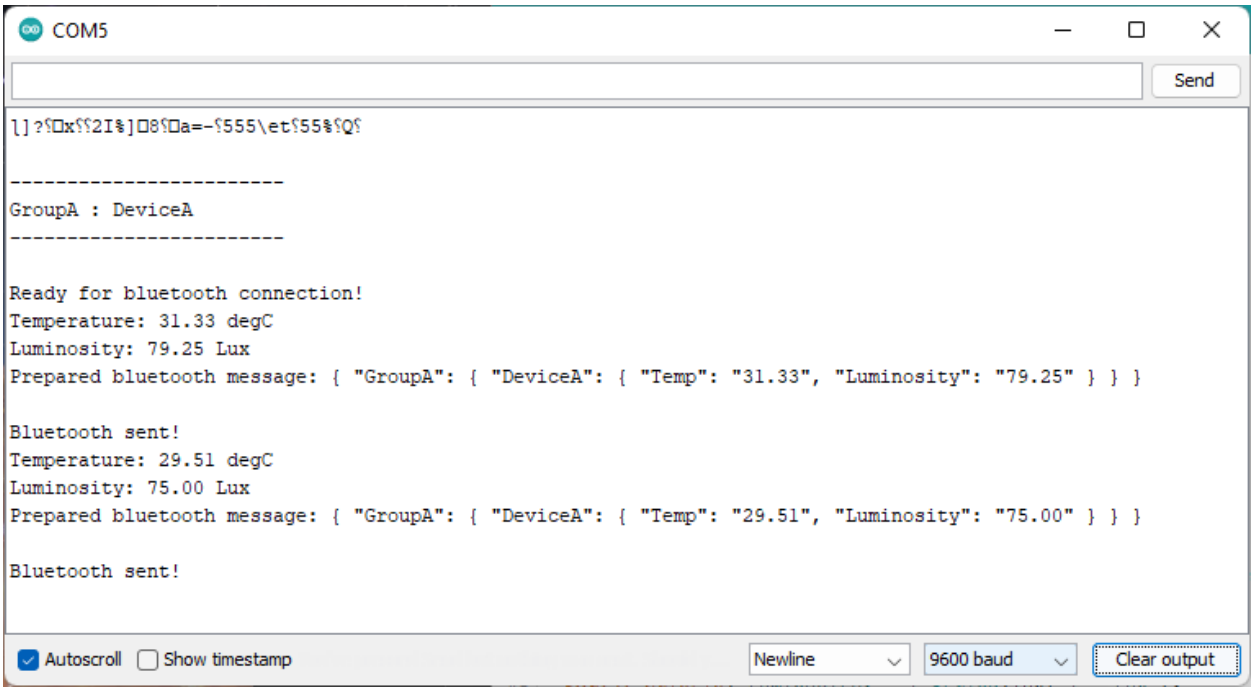
In the **loop()** function of the implementation file, concatenate a JSON object with your desired data. This JSON String is then interpreted as a C-style character array and pushed into the Bluetooth output buffer one element at a time.

Lab02B | Lab02B.h

```
void loop() {

  //Temp sensor
  Wire.beginTransmission(ADDR);
  Wire.write(TMP_CMD);
  Wire.endTransmission();
  delay(100);

  Wire.requestFrom(ADDR, 2);

  char data[2];
  if(Wire.available() == 2){
    data[0] = Wire.read();
    data[1] = Wire.read();
  }

  float temp = ((data[0] * 256.0) + data[1]);
  float tempC = ((175.72 * temp) / 65536.0) - 46.85;
  Serial.println("Temperature: " + String(tempC) + " degC");

  //Sample light sensor
  AsyncAPDS9306Data lightData = lightSensor.syncLuminosityMeasurement();

  //Calculate luminosity
  float lux = lightData.calculateLux();
  Serial.println("Luminosity: " + String(lux) + " Lux");


  //Format data as a JSON string
  String sendData = "{ \"" + groupName + "\": { \"" + deviceName + "\": { \"Temp\": \""
      + String(tempC) + "\", \"Luminosity\": \"" + String(lux) + "\" } } }" + '\n';

  Serial.println("Prepared bluetooth message: " + sendData);

  //Push the string characters onto the bluetooth output buffer
  for (int i = 0; i < strlen(sendData.c_str()) + 1; i++){
    SerialBT.write(sendData.c_str()[i]);
  }

  delay(500);
  Serial.println("Bluetooth sent!");

  delay(DELAY_BETWEEN_SAMPLES_MS);

}
```

Upload the modified project to your MacIoT board.

Open **Serial Monitor** and observe the serial output.


COM5
Send
]]?¶□x¶¶2I%]□8¶□a=-¶555\et¶55%¶Q¶

GroupA : DeviceA

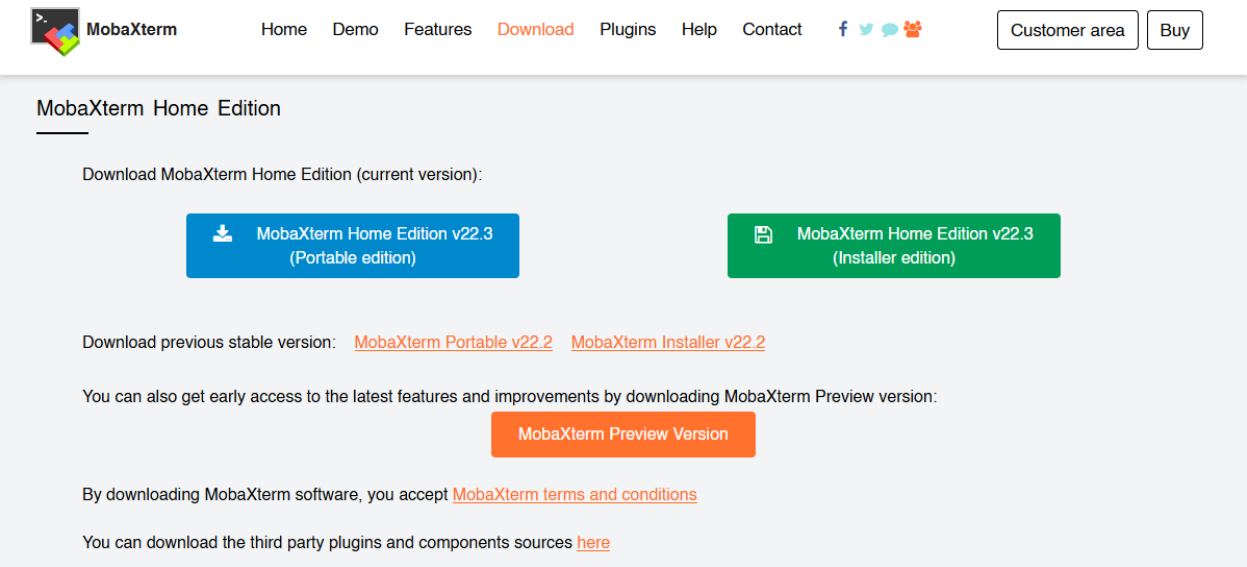
Ready for bluetooth connection!
Temperature: 31.33 degC
Luminosity: 79.25 Lux
Prepared bluetooth message: { "GroupA": { "DeviceA": { "Temp": "31.33", "Luminosity": "79.25" } } }
Bluetooth sent!
Temperature: 29.51 degC
Luminosity: 75.00 Lux
Prepared bluetooth message: { "GroupA": { "DeviceA": { "Temp": "29.51", "Luminosity": "75.00" } } }
Bluetooth sent!
Autoscroll
Show timestamp
Newline
9600 baud
Clear output

Viewing a Bluetooth Serial Device

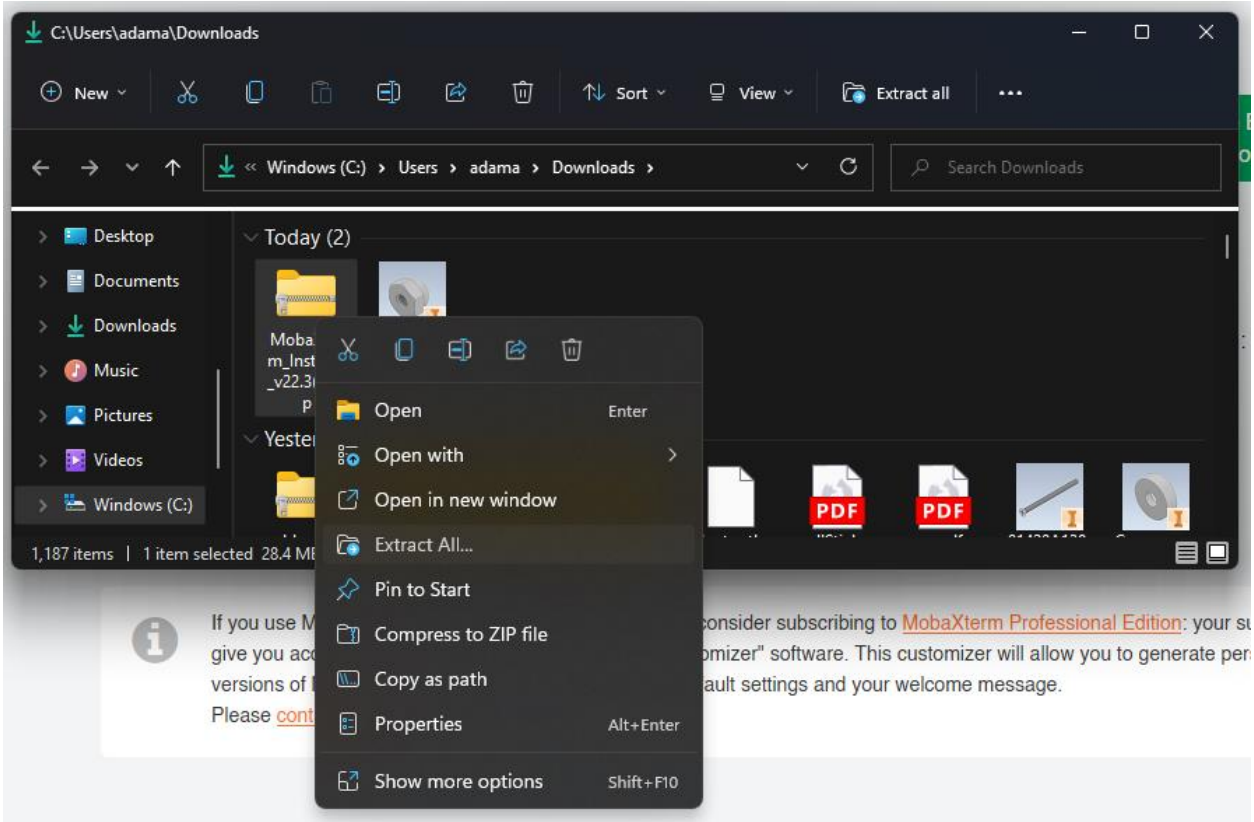
Navigate to the following URL and install **MobaXTerm Home Edition**.

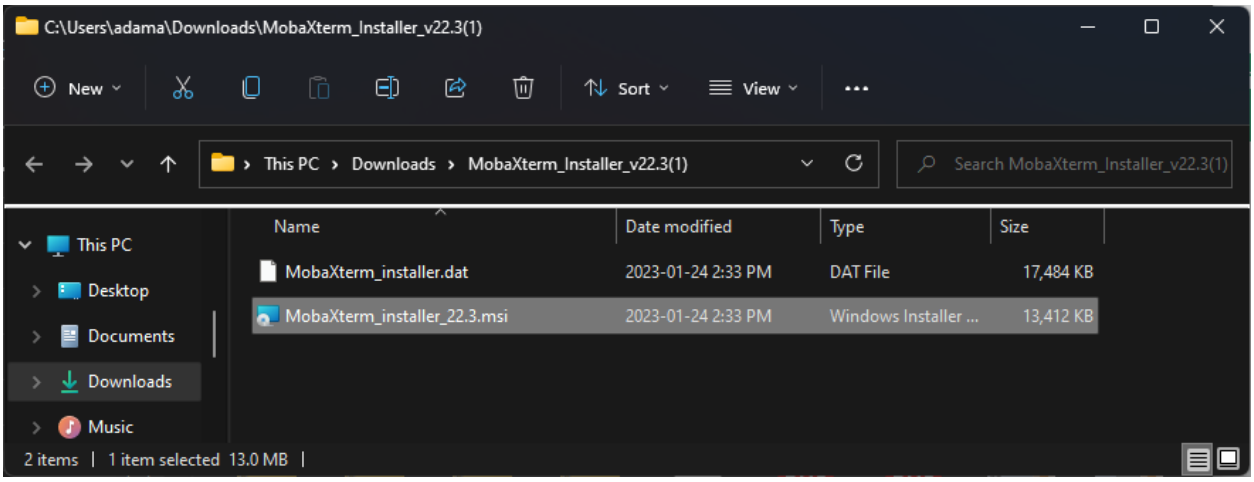
https://mobaxterm.mobatek.net/download-home-edition.html

Choose the **Installer Edition**.

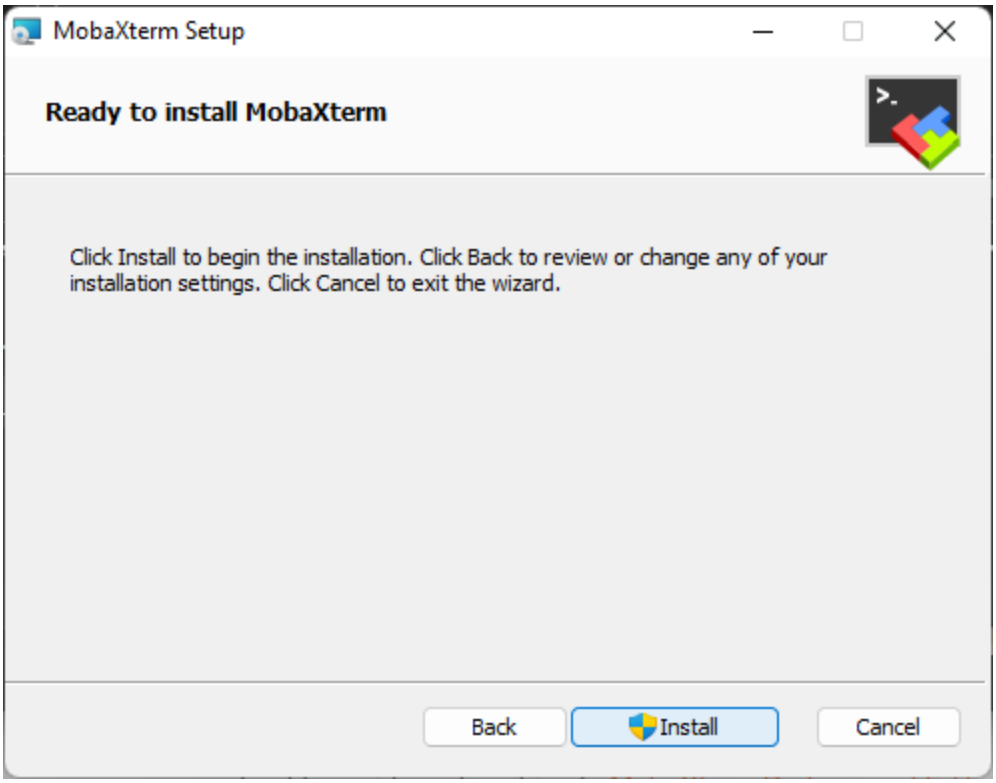


A **ZIP** file will be downloaded. This file must be extracted and the **MSI** installer must be ran as **Administrator**.

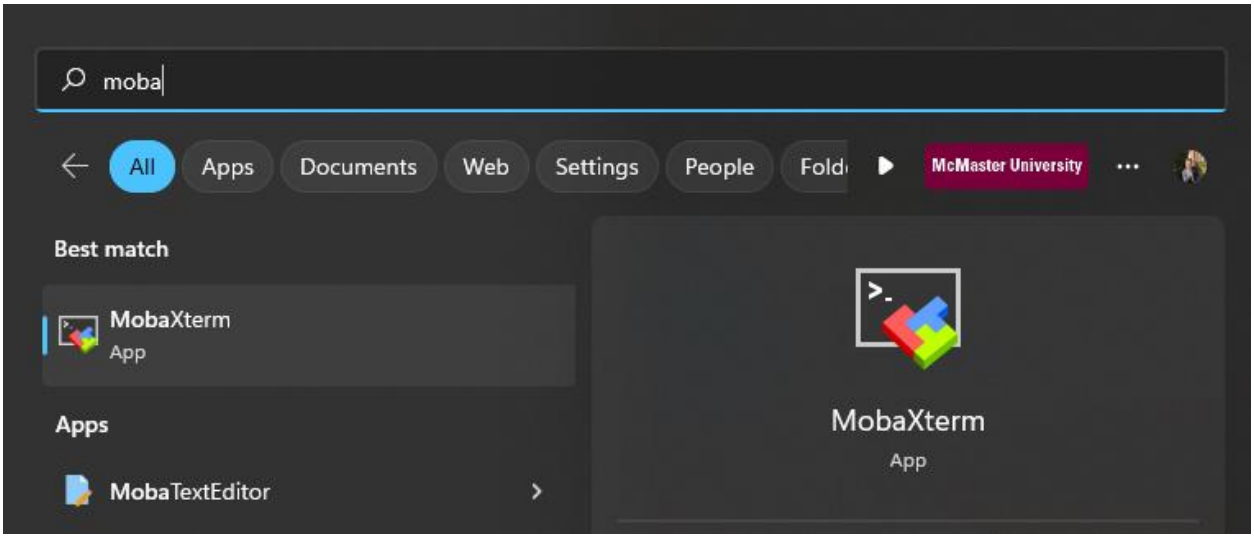




Install using the default options.

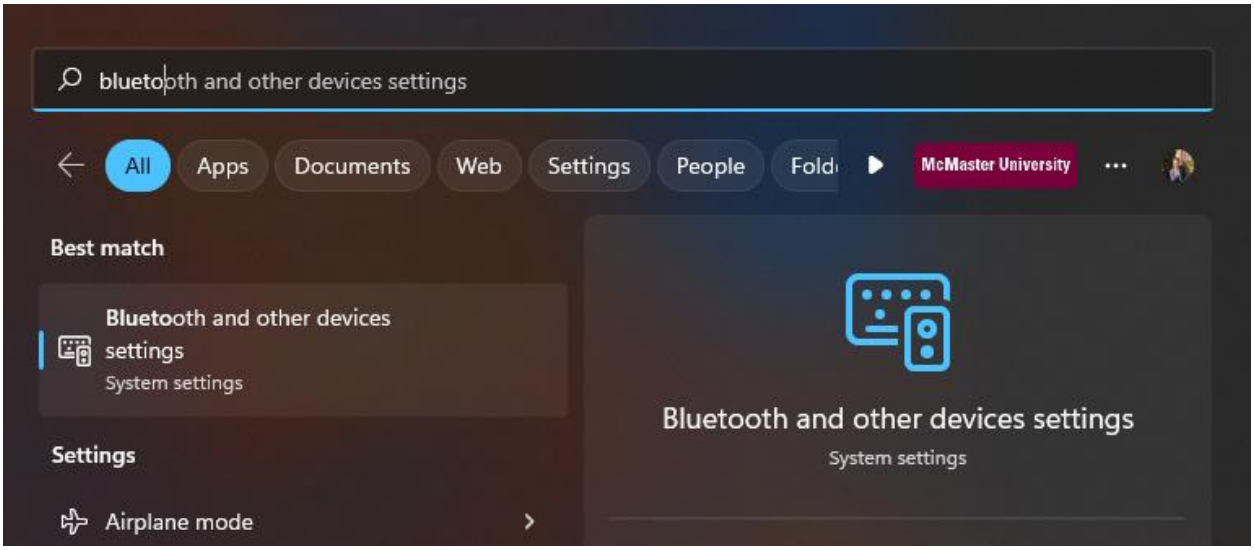


Launch **MobaXTerm**.

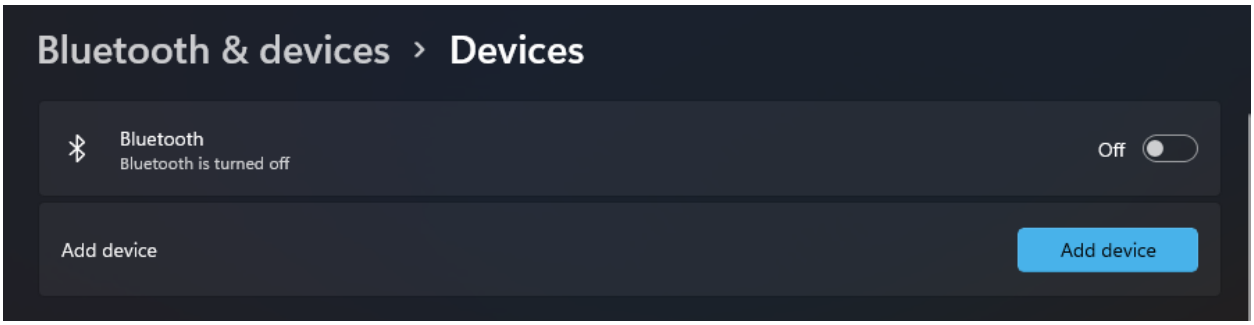


MobaXTerm is a feature filled application that is used to manage servers and ports. We will be using it to open the Bluetooth serial port and ensure that data is being transmitted between the MacIoT board and our PC.

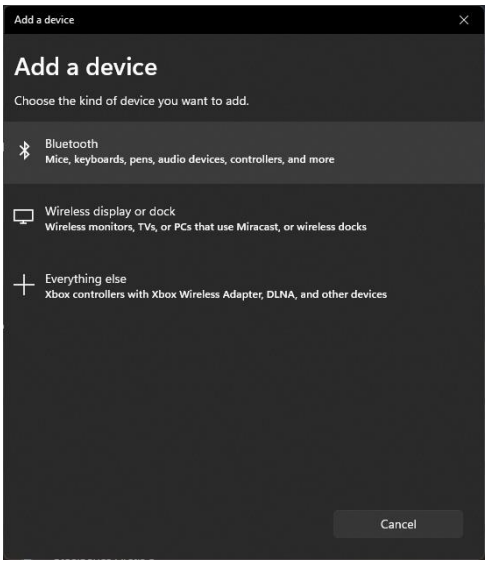
Before proceeding, enable Bluetooth on your PC and connect to the microcontroller.



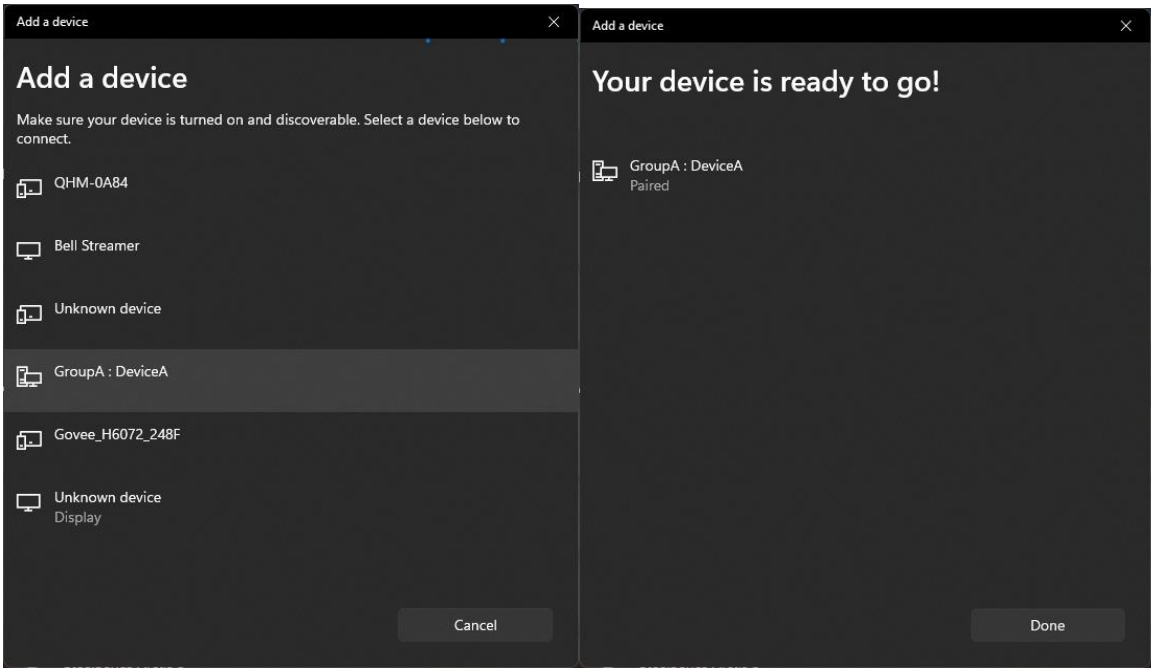
Toggle Bluetooth **On** and press **Add device**.



Choose **Bluetooth**.



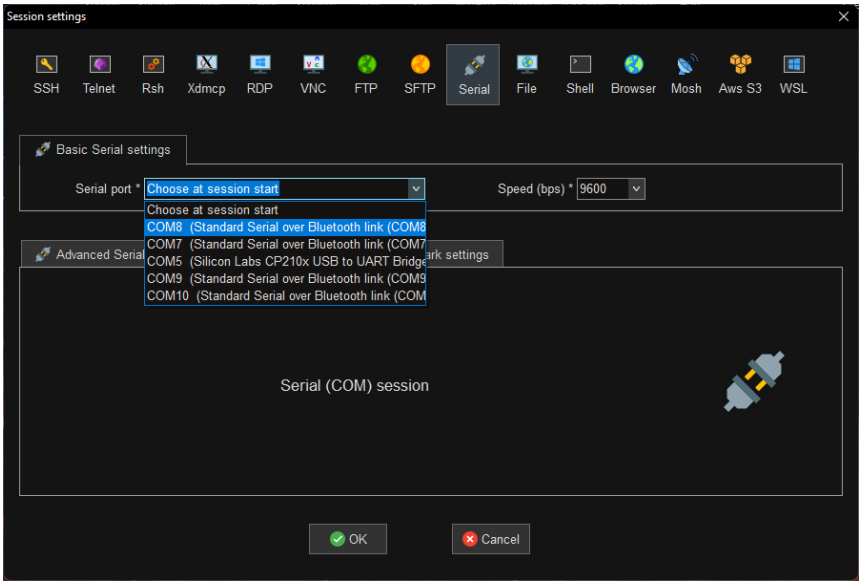
Press on your microcontroller to connect.



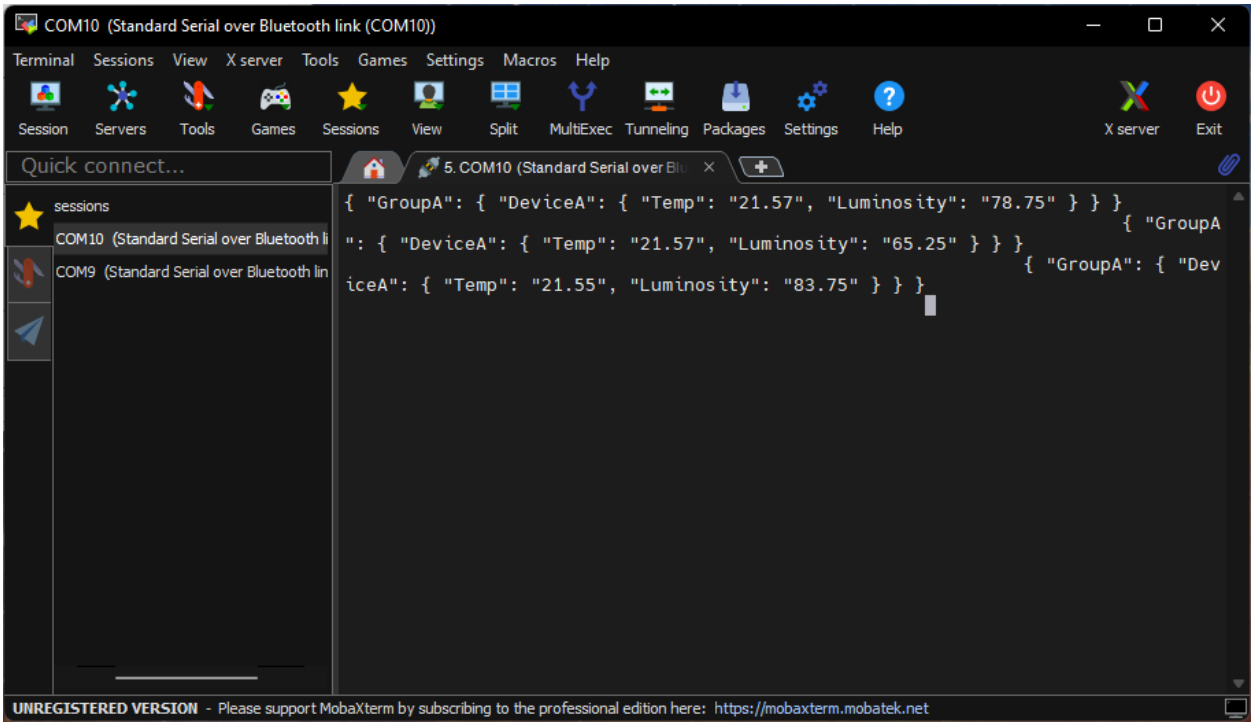
Navigate back to **MobaXTerm** and press **Session**.

Select **Serial** from the top ribbon.

Choose a Bluetooth serial port. You may need to try all that are available to find which one is transmitting.



In my case, this ended up being **COM10** at **9600 baud**.



Exercise A

Modify the MacIoT code so that **Humidity** is read from the sensor, formatted as a **JSON attribute**, and transmitted to MQTT.

NOTE: The Python routing script will automatically parse the new parameter.

IIC Humidity Command = 0xF5

IIC Humidity Address = 0x40 (same as temp)

What is JSON?: https://www.w3schools.com/js/js_json_syntax.asp

Parsing Data using Python

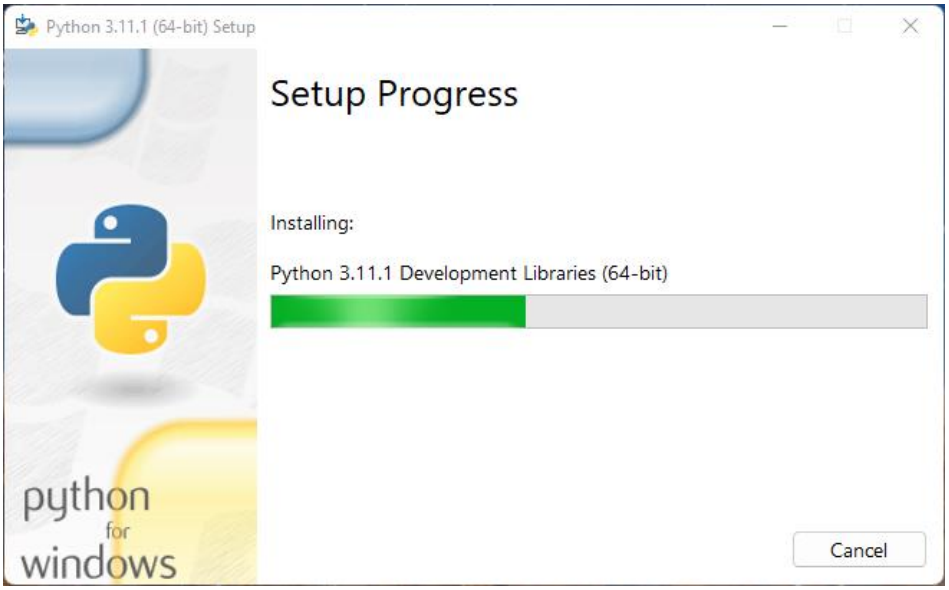
It may not be enough to view the data. We want to be able to act on the data being sent, collect it over time, and visualize it on an HMI or familiar format.

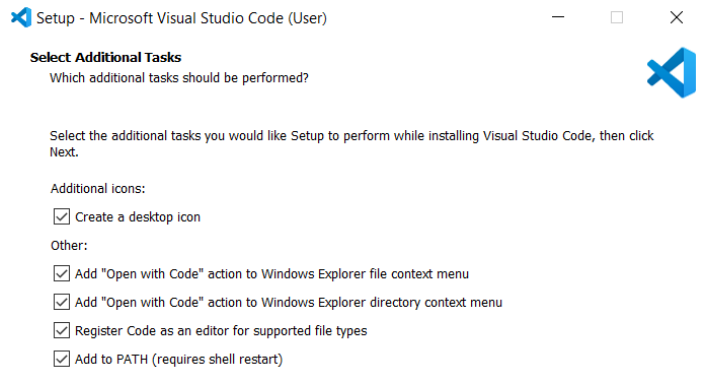
Installing Python

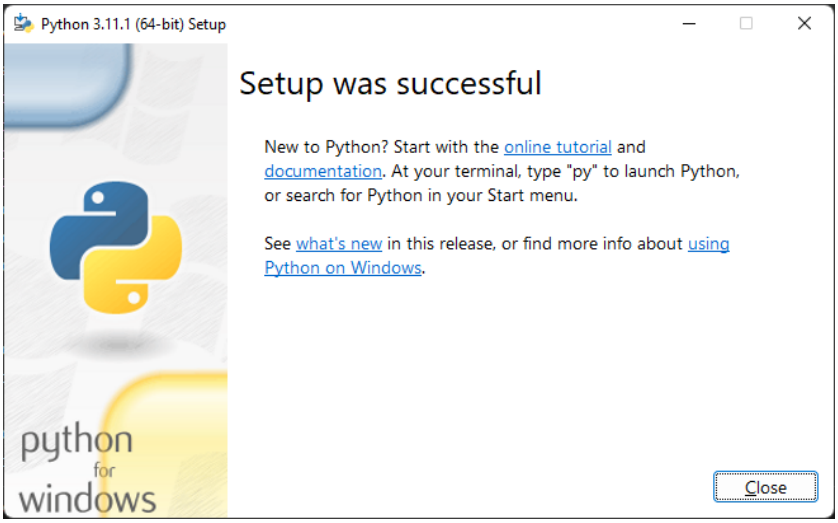
Navigate to the following website and install the latest version of Python 3.

https://www.python.org/downloads/

Install using the default options. If there is a checkbox to **Add to Path**, please **check this checkbox**.





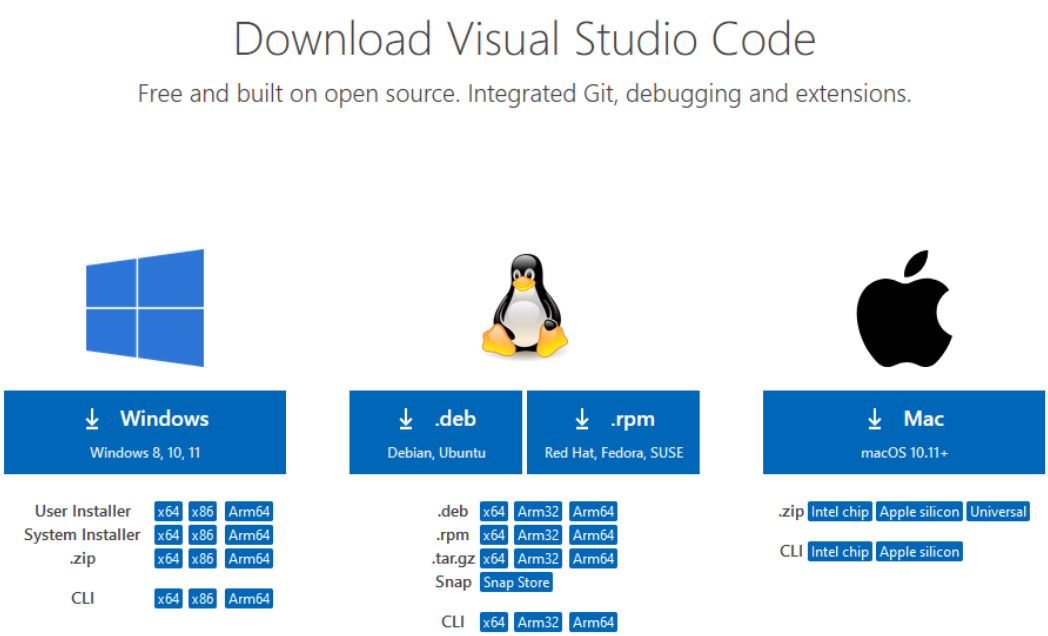


Once Python is installed, please **restart** your computer.

Installing VSCode

Navigate to the following URL:

https://code.visualstudio.com/download



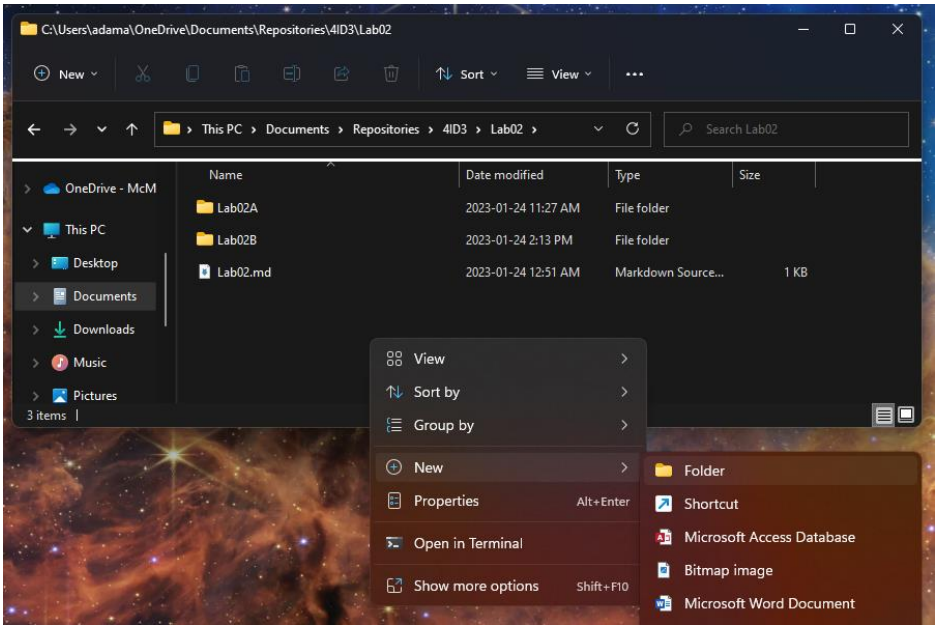
Run the installer using the default install options.

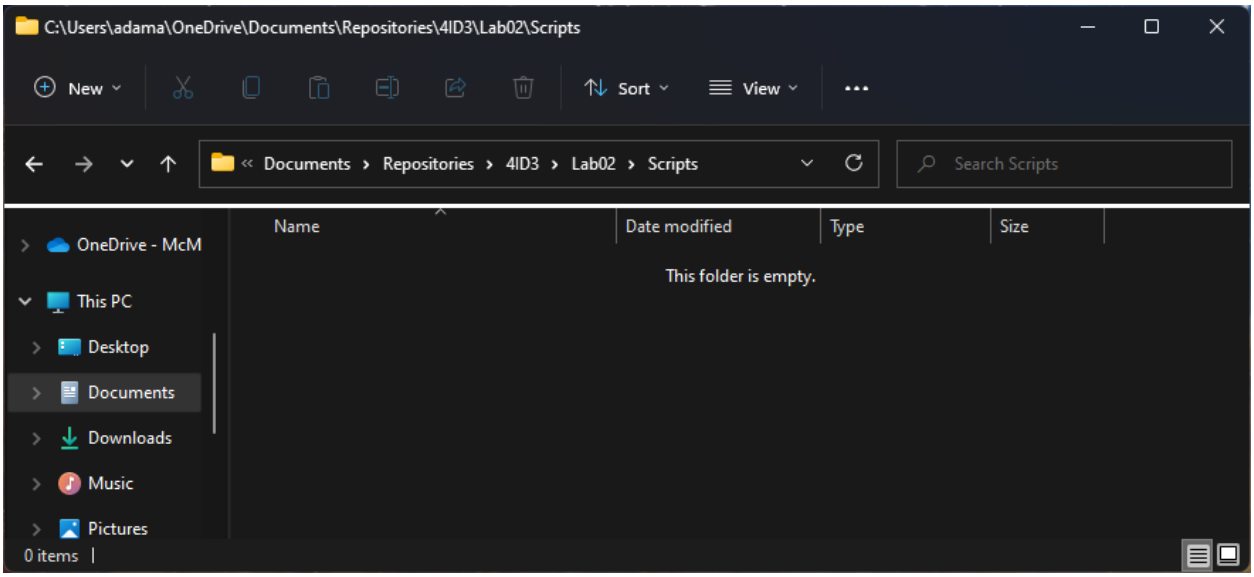
Once installed launch VSCode.

Open the **Extensions** tab using the ribbon on the left-side.

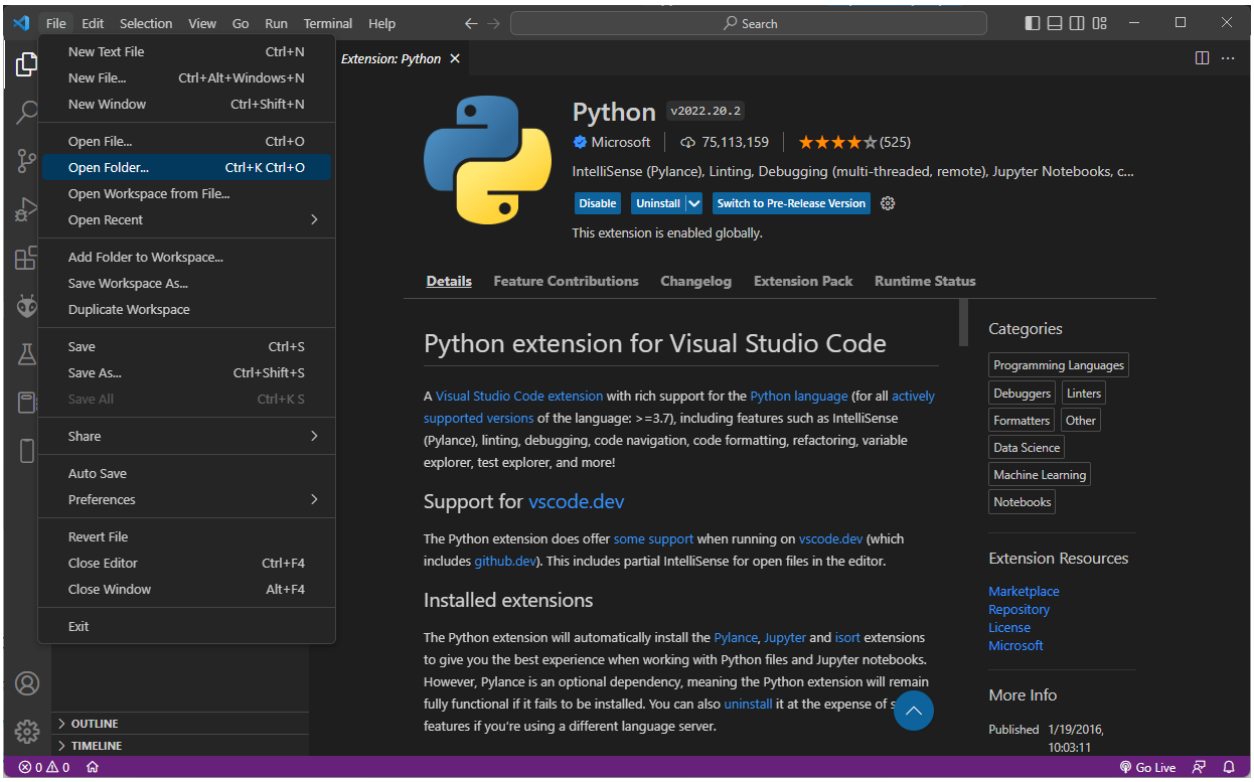
Search for **Python** and **Install** the one provided by **Microsoft**.

Within the **Lab02** folder, create a new folder named **Scripts**.

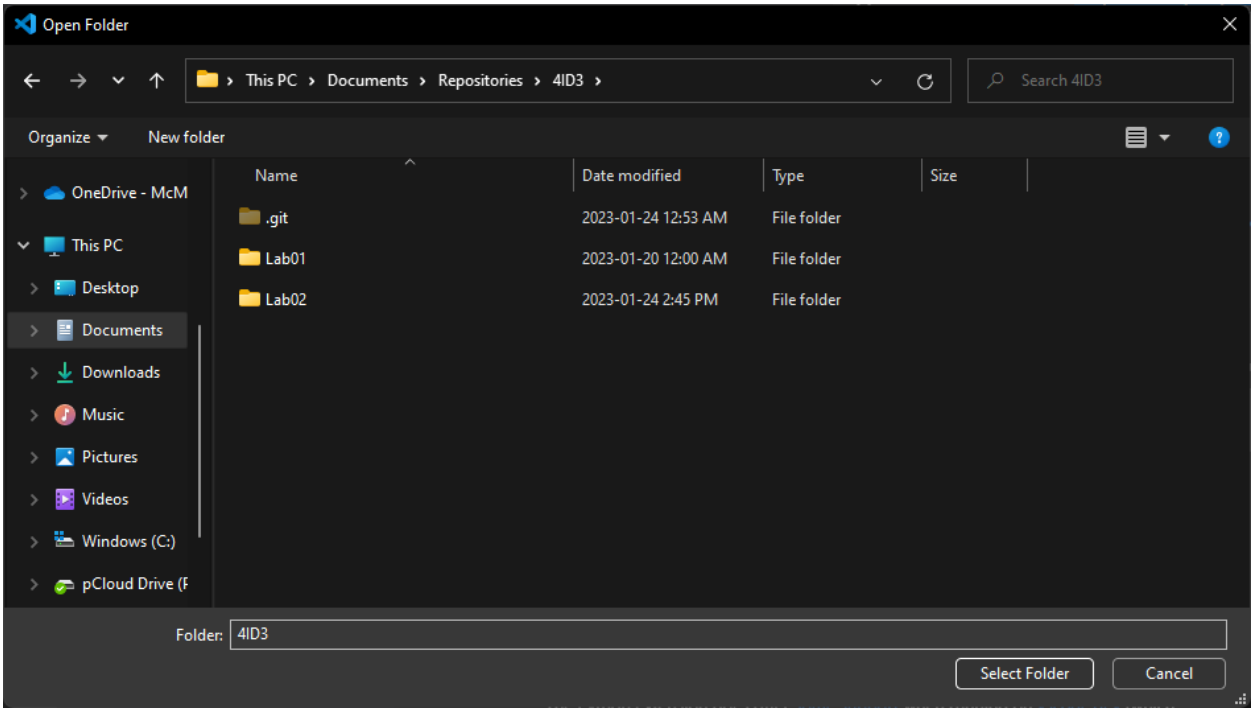




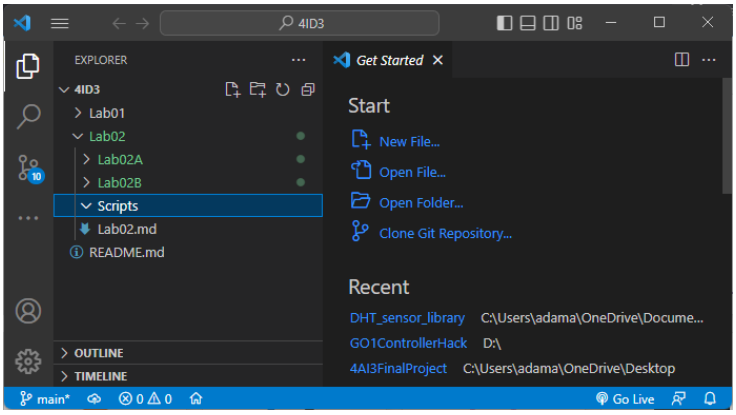
Within VSCode, do **File > Open Folder**.



Open your local repository.



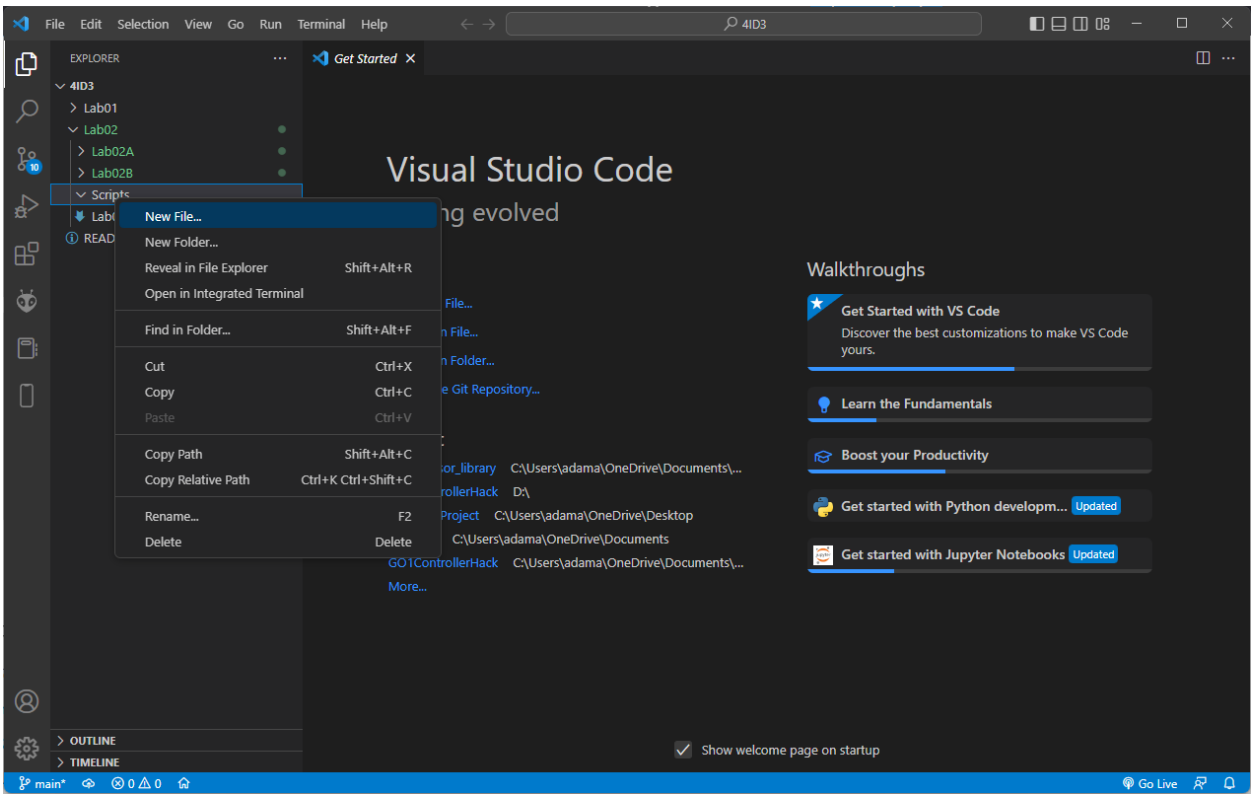
You should be able to see your **Scripts/** folder in the **File Explorer** on the left-tab of your **VSCode Editor**.



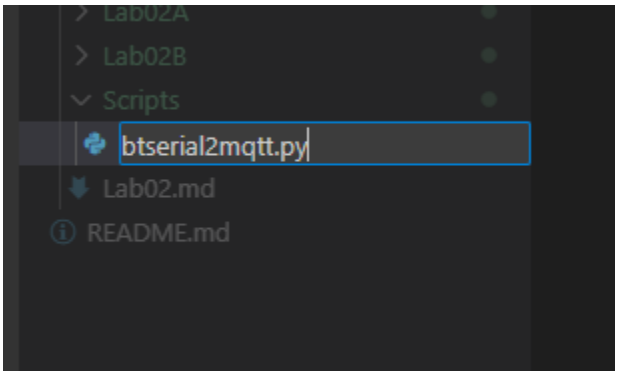
MQTT Routing

The first script that we will write will parse the JSON string with our smart device data and route it to its appropriate MQTT path.

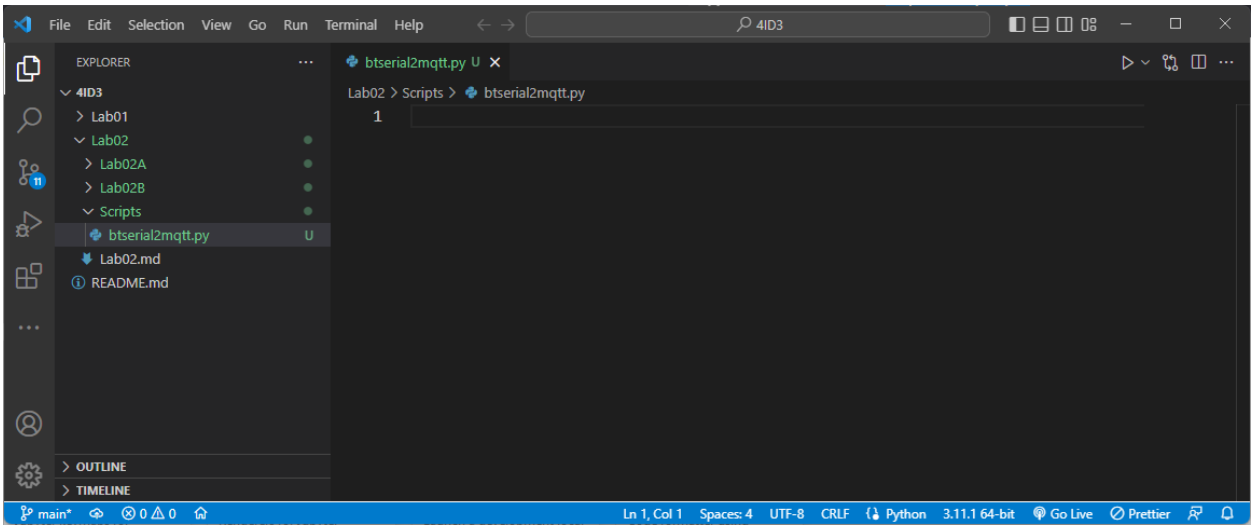
Right-click and select **New File**.



Name it **btserial2mqtt.py**.



Double-click to open it in the text editor.



Copy out the following python script:

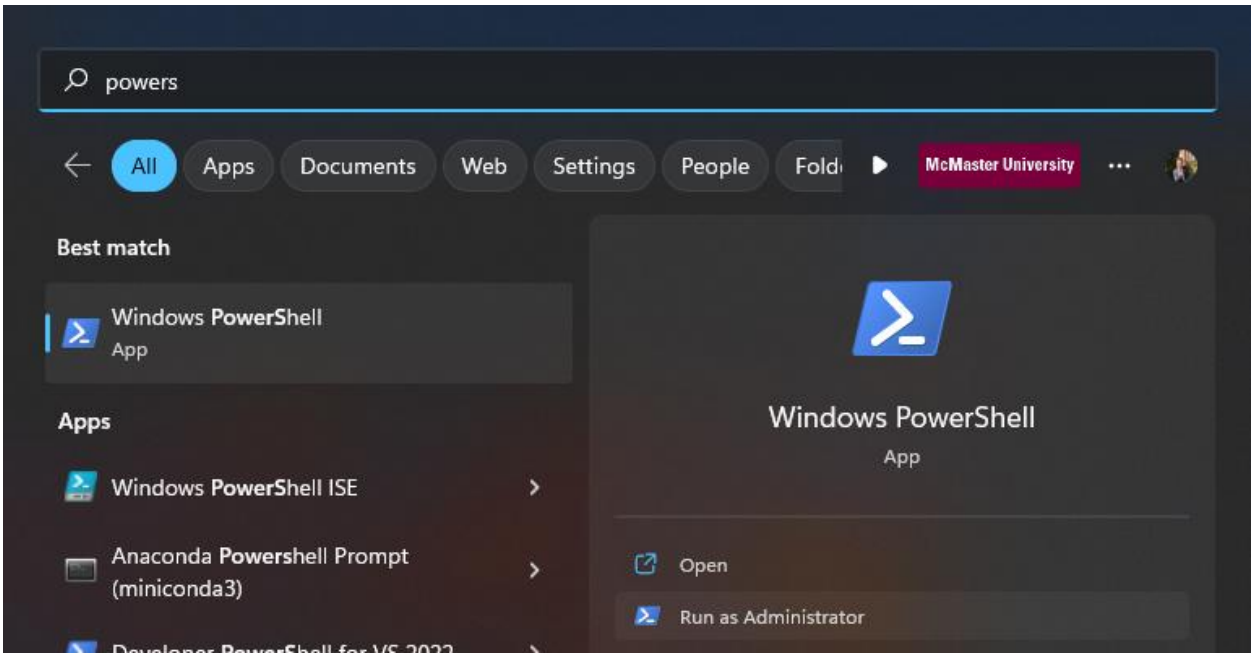
```
import serial
import json
import paho.mqtt.client as mqtt
import time
mqttIp = None
mqttPort = None
bluetoothCOM = None


mqttIp = input("MQTT Broker IP: ")
if(mqttIp == None or mqttIp == ''): mqttIp = 'test.mosquitto.org'

mqttPort = input("MQTT Broker Port: ")
if(mqttPort == None or mqttPort== ''): mqttPort = 1883

bluetoothCOM = input("Bluetooth COM (e.g. COM7): ")
if(bluetoothCOM == None or bluetoothCOM  == ''): bluetoothCOM  = 'COM7'
```

```
print(f'\n-------\nCONFIGURATION\n-------\nIP: {mqttIp}\nPORT: {mqttPort}\nBT COM: {bluetoothCOM}')

def on_connect(client, userdata, flags, rc):
   print("Connected to  "+str(rc))

def on_message(client, userdata, msg):
   print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
print("Connecting to MQTT")
for x in range(7):
   print(".")
   time.sleep(0.7)
client.connect(mqttIp, mqttPort, 60)

print("Connecting to serial: " + bluetoothCOM)
time.sleep(1)
try:
   ser = serial.Serial(bluetoothCOM, 9600)
   print("Bluetooth COM opened")
except:
   exit(1)

while True:
   client.loop()
   cc=str(ser.readline())
   #cc = cc[6:][:-3]
   firstSplitIndex = cc.find('{')
   secondSplitIndex = cc.rfind('}')
   cc = cc[firstSplitIndex: secondSplitIndex+1]
   #print(cc)


   try:
      jDict = json.loads(cc)
      #print(jDict)
      groupName = list(jDict.keys())[0]
      #print(groupName)
      deviceId = list(jDict[groupName])[0]
      #print(deviceId)
      print(f'DeviceID: {deviceId} -> {jDict[groupName][deviceId]}')
      for key, val in jDict[groupName][deviceId].items():
         print(f'{groupName}/{deviceId}/{key} -> {val}')
         client.publish(f'{groupName}/{deviceId}/{key}', val.encode("UTF-8"))
         #client.wait_for_publish()

   except:
      print("Failed to decode: ")
      print(cc)

client.loop_forever()
```

```
ser.close()
```

For the above script to work, we must install 2 libraries.

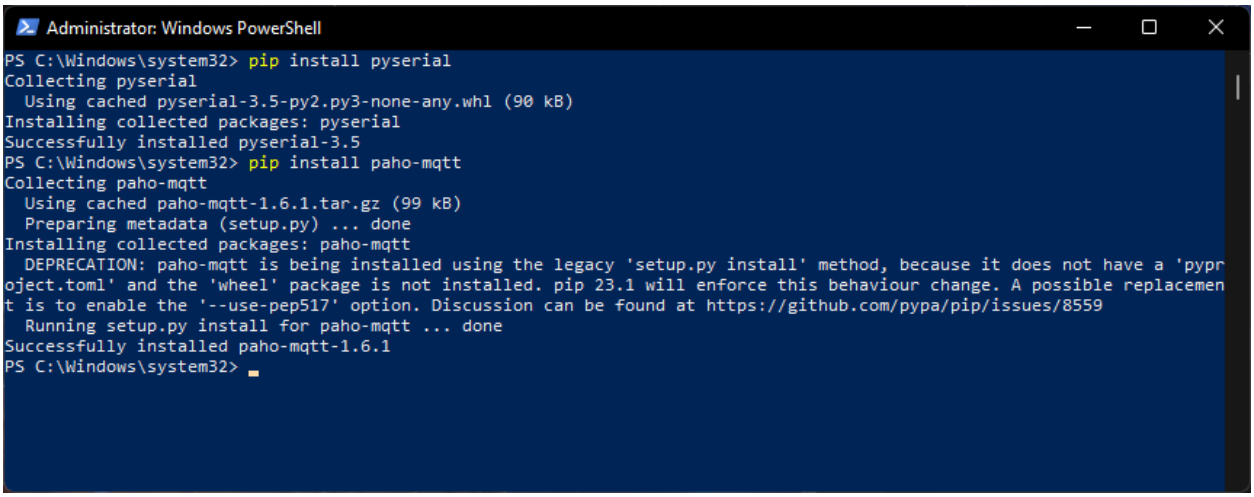
Open **Powershell** as **Administrator**.



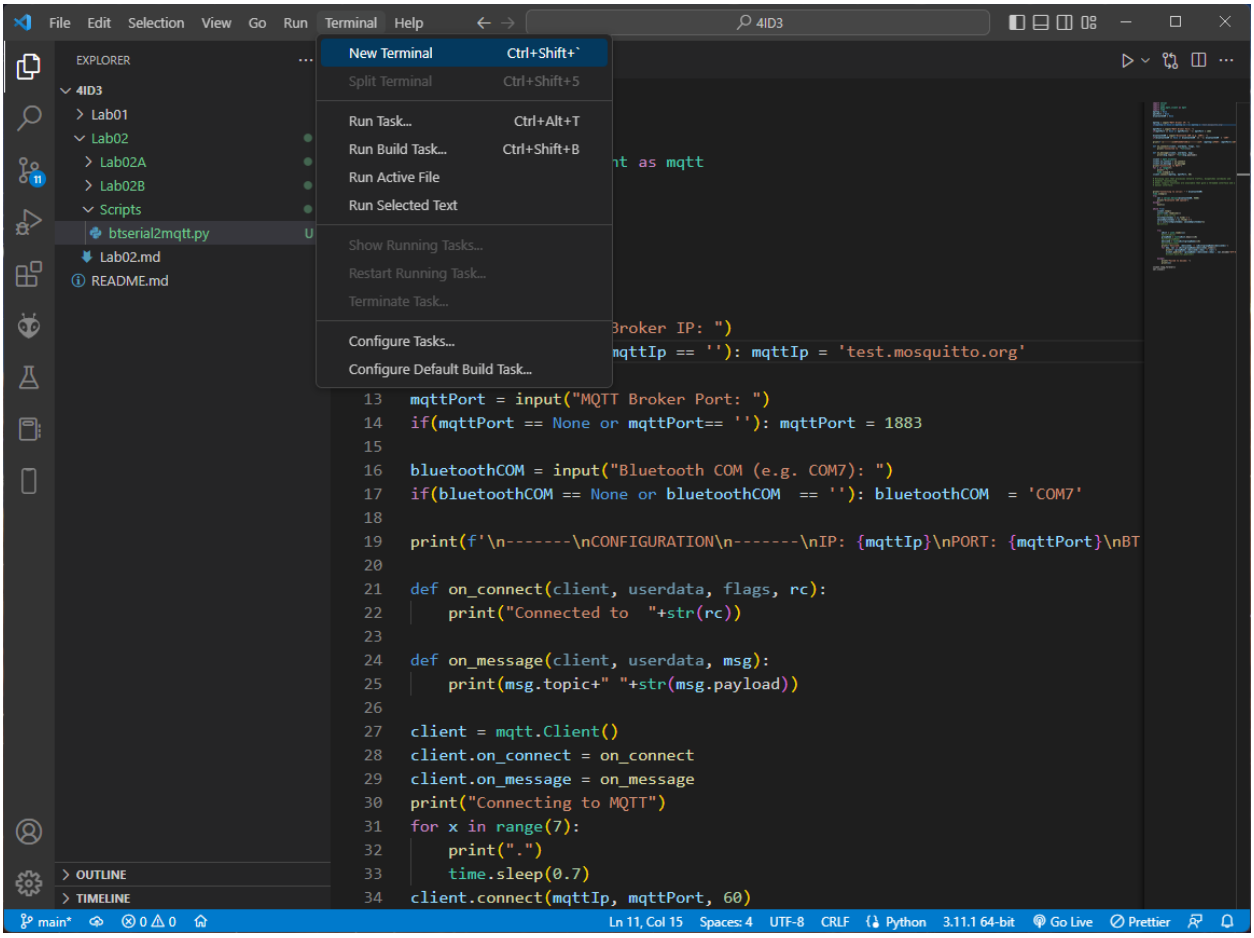
Run the following two commands:

pip install pyserial

pip install paho-mqtt

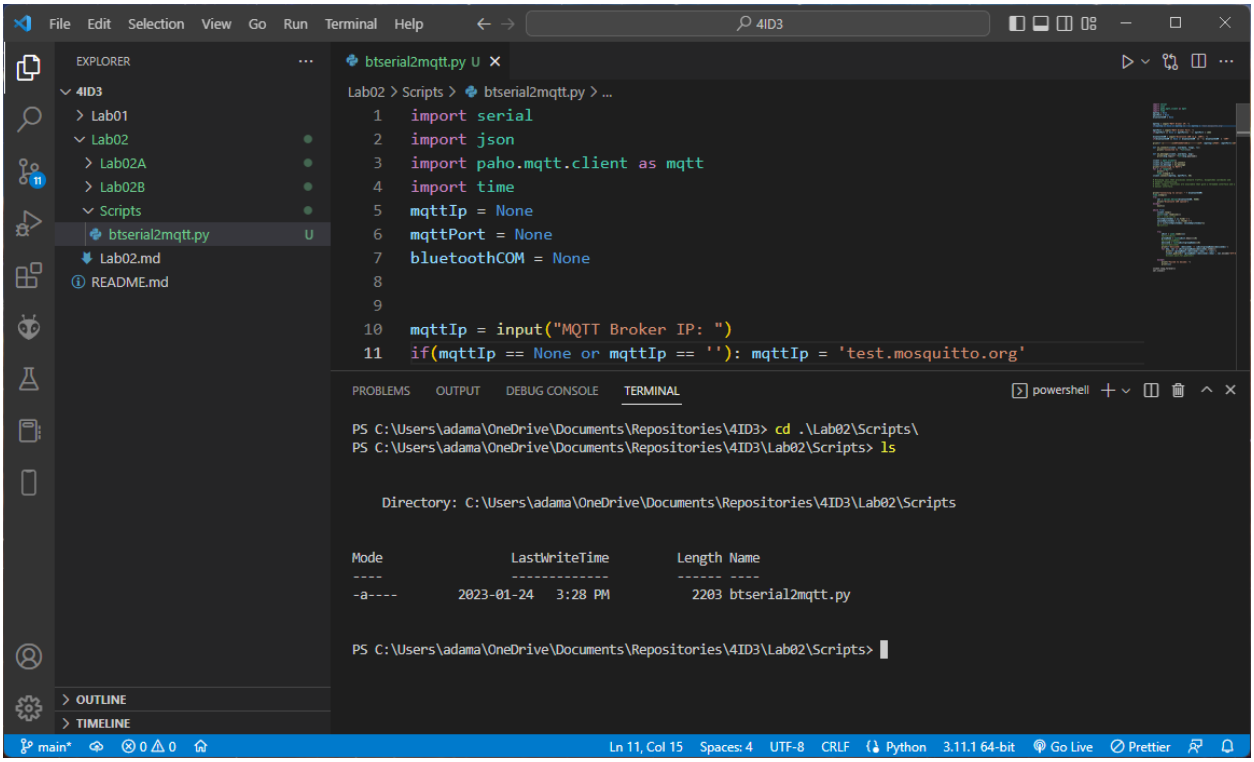


Within VSCode, navigate to **Terminal > New Terminal**.



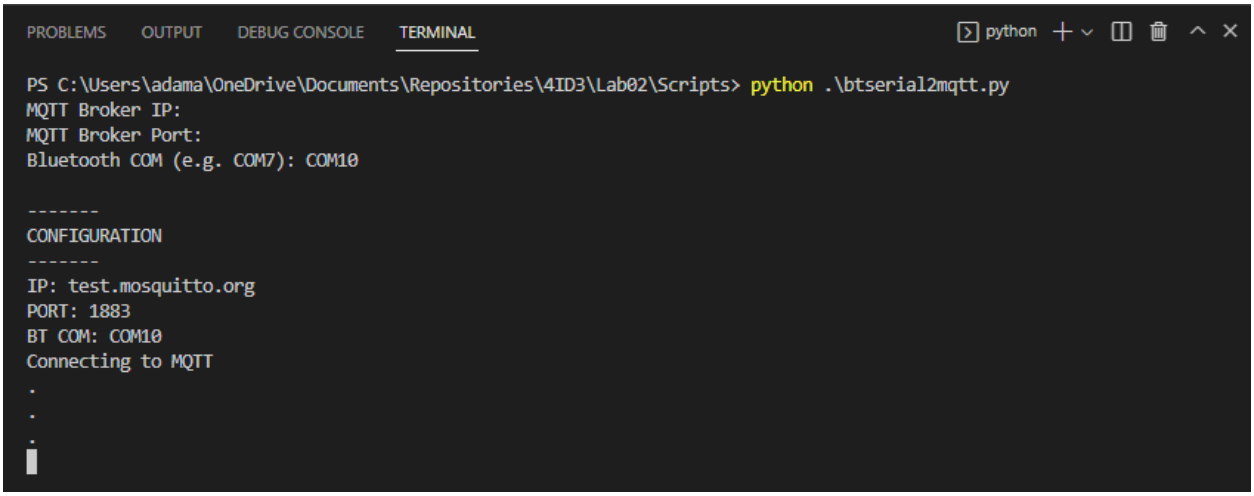
A new Powershell terminal window will open inside VSCode.

Navigate to your **Lab02/Scripts/** directory.



To launch the python application, run the following command:

python ./btserial2mqtt.py



Fill in the prompted information. Press **enter** to use test.mosquitto.org defaults.

Ensure that your MacIoT board is still transmitting. Press **RESET** to restart it. Wait 10 seconds.

ENSURE THAT NO OTHER BLUETOOTH SERIAL MONITORS ARE OPEN, BLOCKING THE PORT.

Run btserial2mqtt.py using the following command:

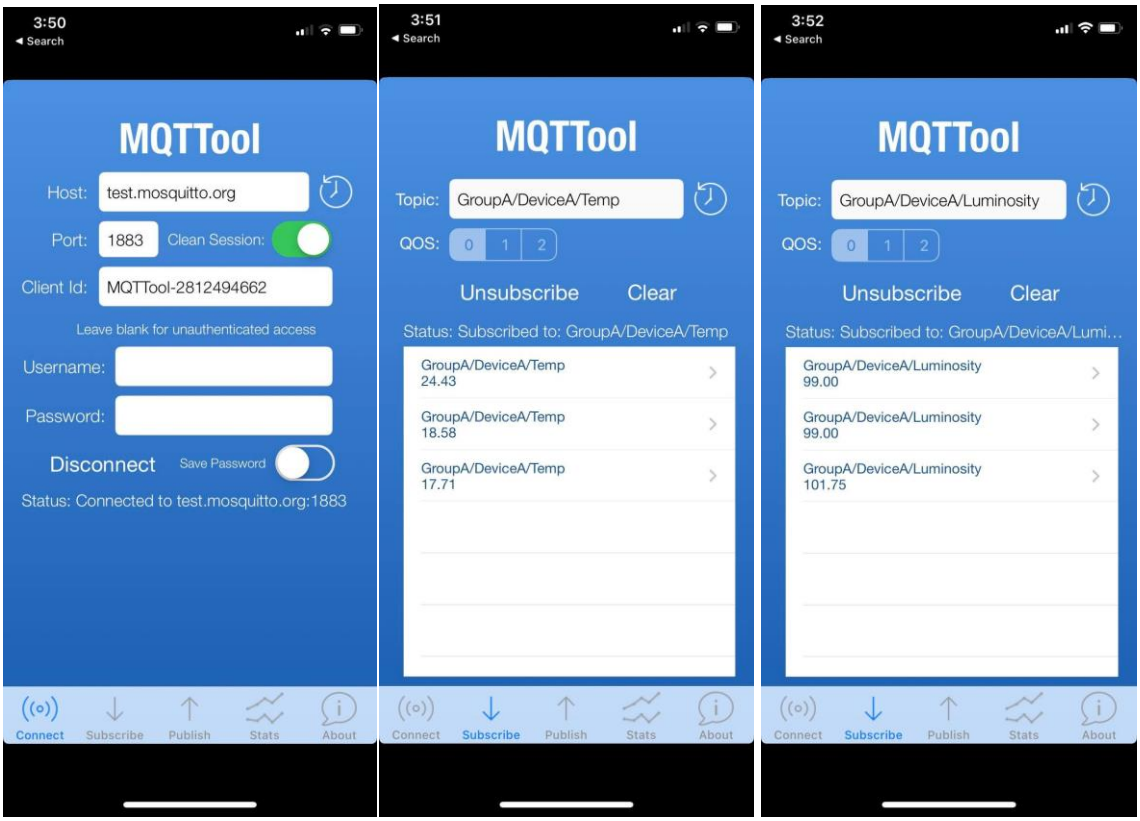
Python ./btserial2mqttt.py

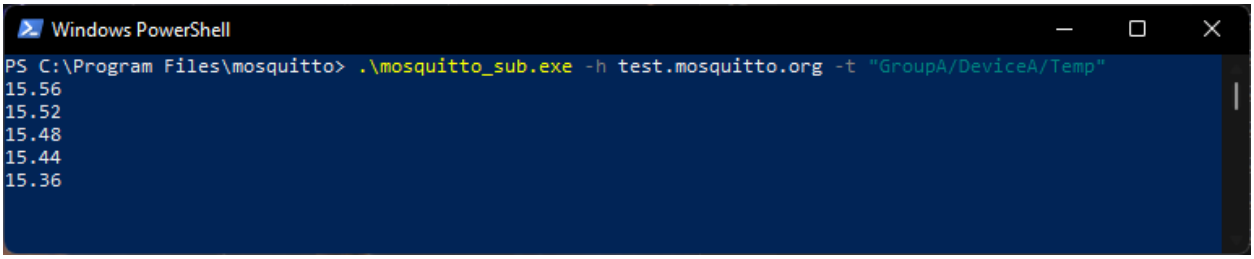
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> python .\btserial2mqtt.py
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

IP: test.mosquitto.org
PORT: 1883
BT COM: COM10
Connecting to MQTT
.
.
.
.
.
.
.
Connecting to serial: COM10
Bluetooth COM opened:  Serial<id=0x282fdb5f370, open=True>(port='COM10', baudrate=9600, bytesize=
8, parity='N', stopbits=1, timeout=None, xonxoff=False, rtscts=False, dsrdtr=False)
Connected to  0
DeviceID: DeviceA -> {'Temp': '17.83', 'Luminosity': '66.75'}
GroupA/DeviceA/Temp -> 17.83
GroupA/DeviceA/Luminosity -> 66.75
```

Verify MQTT

Use the MQTT application on your phone or the Mosquitto MQTT client installed previously to verify that data is being transmitted to the correct paths.



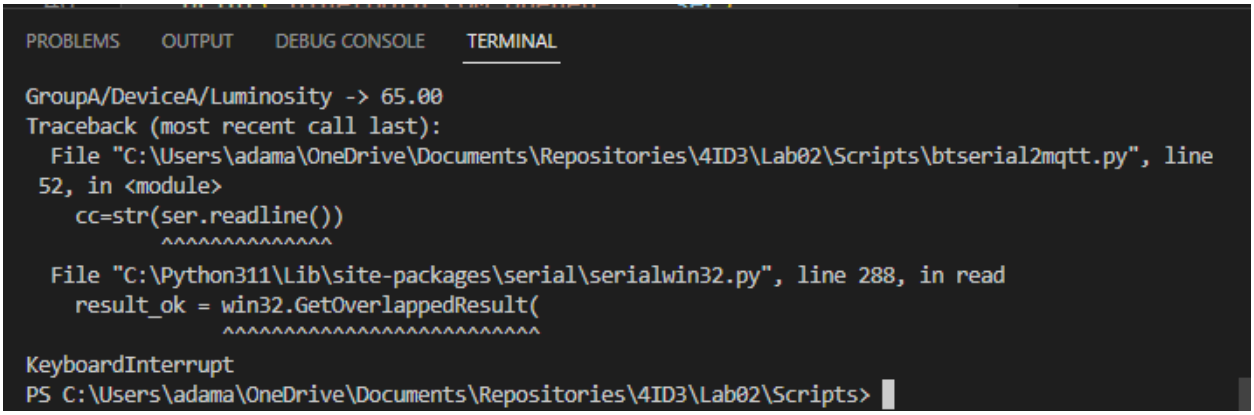


Exercise B

Now that data is being transmitted to MQTT, using knowledge from previous labs, set up a **NodeRED flow** to **read** this data from the public MQTT broker and **visualize** it.

Exiting the Python Application

Press **CTRL + C** to exit the Python application.

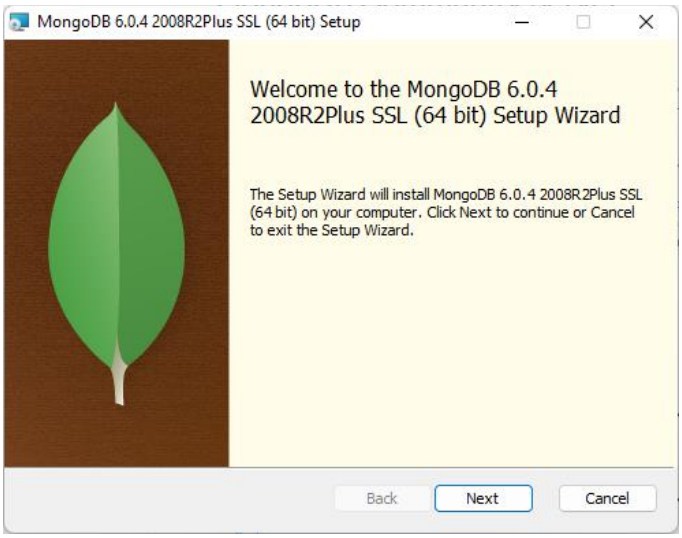

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
GroupA/DeviceA/Luminosity -> 65.00
Traceback (most recent call last):
File "C:\Users\adama\OneDrive\Documents\Repositories\4ID3\Lab02\Scripts\btserial2mqtt.py", line 52, in <module>
cc=str(ser.readline())
^^^^^^^^^^^^^^
File "C:\Python311\Lib\site-packages\serial\serialwin32.py", line 288, in read
result_ok = win32.GetOverlappedResult(
^^^^^^^^^^^^^^^^^^^^^^^^^^^
KeyboardInterrupt
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3\Lab02\Scripts>

Optional: Connecting to a Database

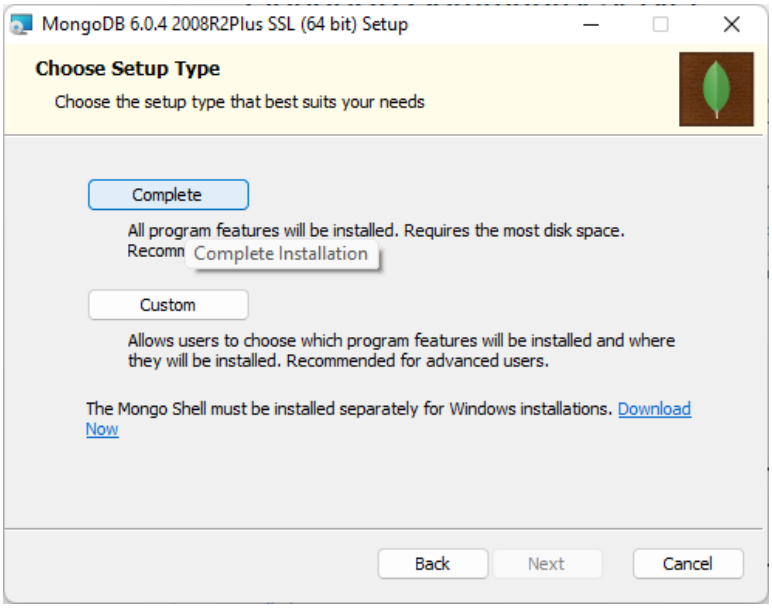
Navigate to the following URL and download **MongoDB Community**.

https://www.mongodb.com/try/download/community-kubernetes-operator

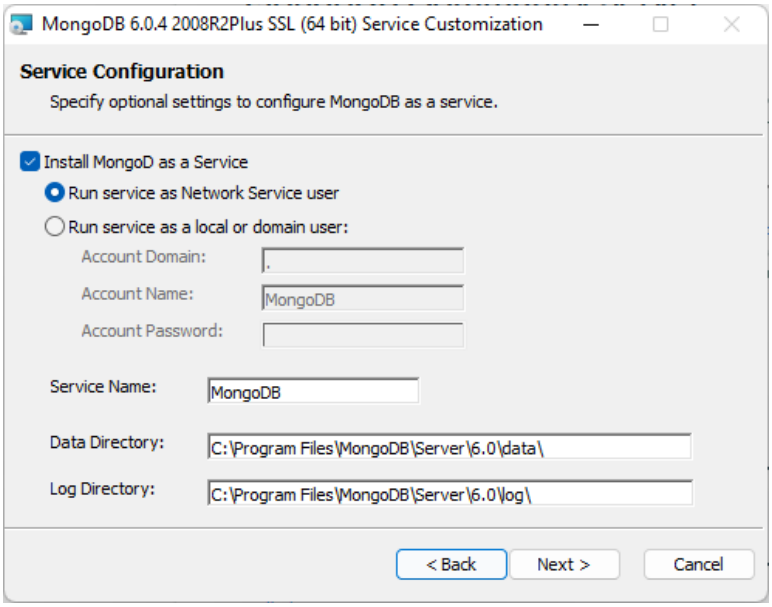
Run the installer using default settings. It will say **This may contain malicious software**. Ignore this warning, this software is very reputable, the packaging format is just out-of-date.



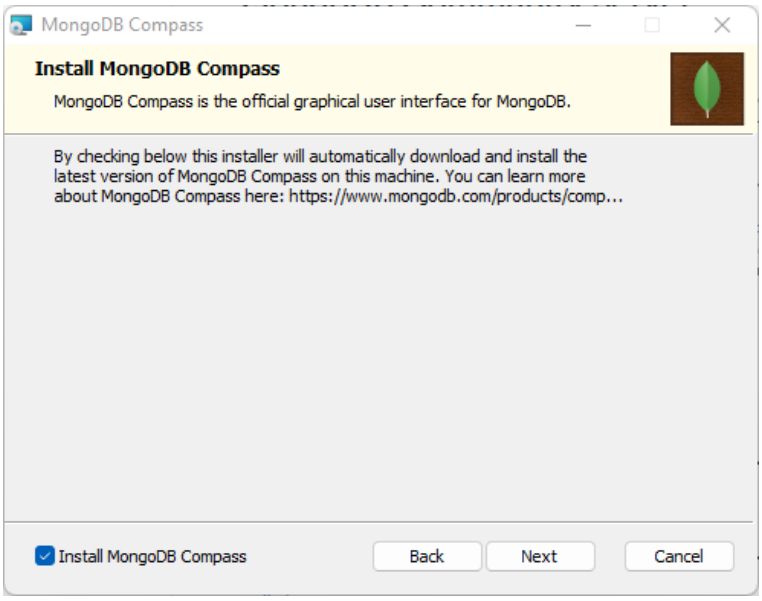
Perform a **Complete** installation.

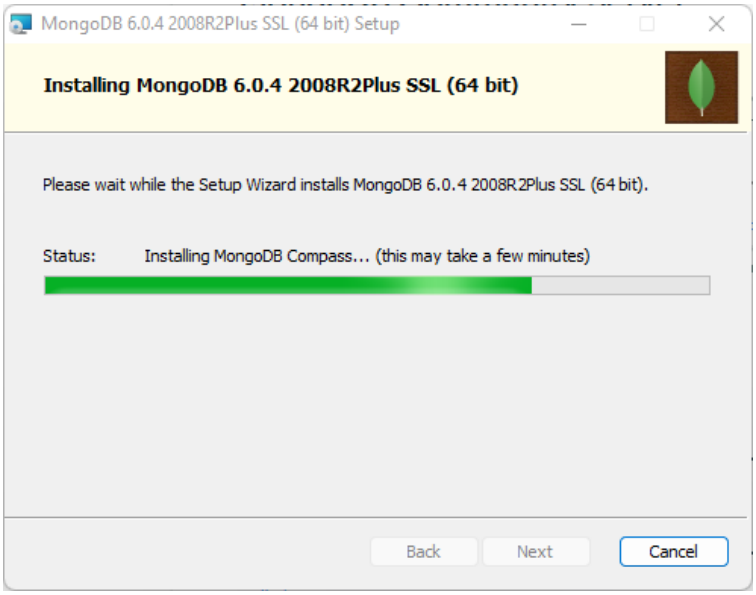


Leave **Service Configuration** as default.

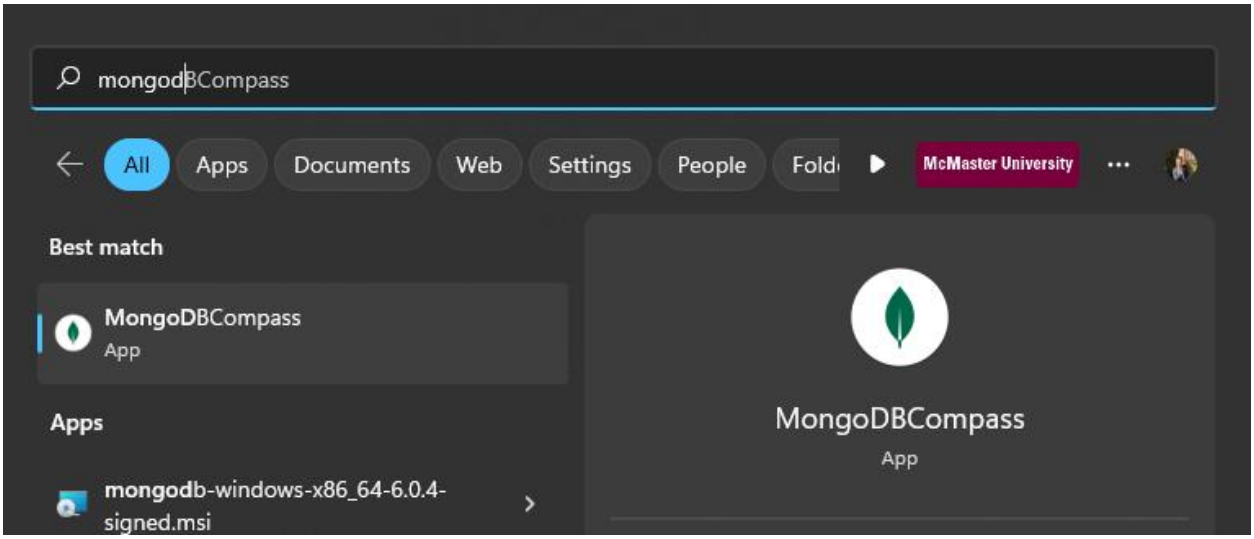


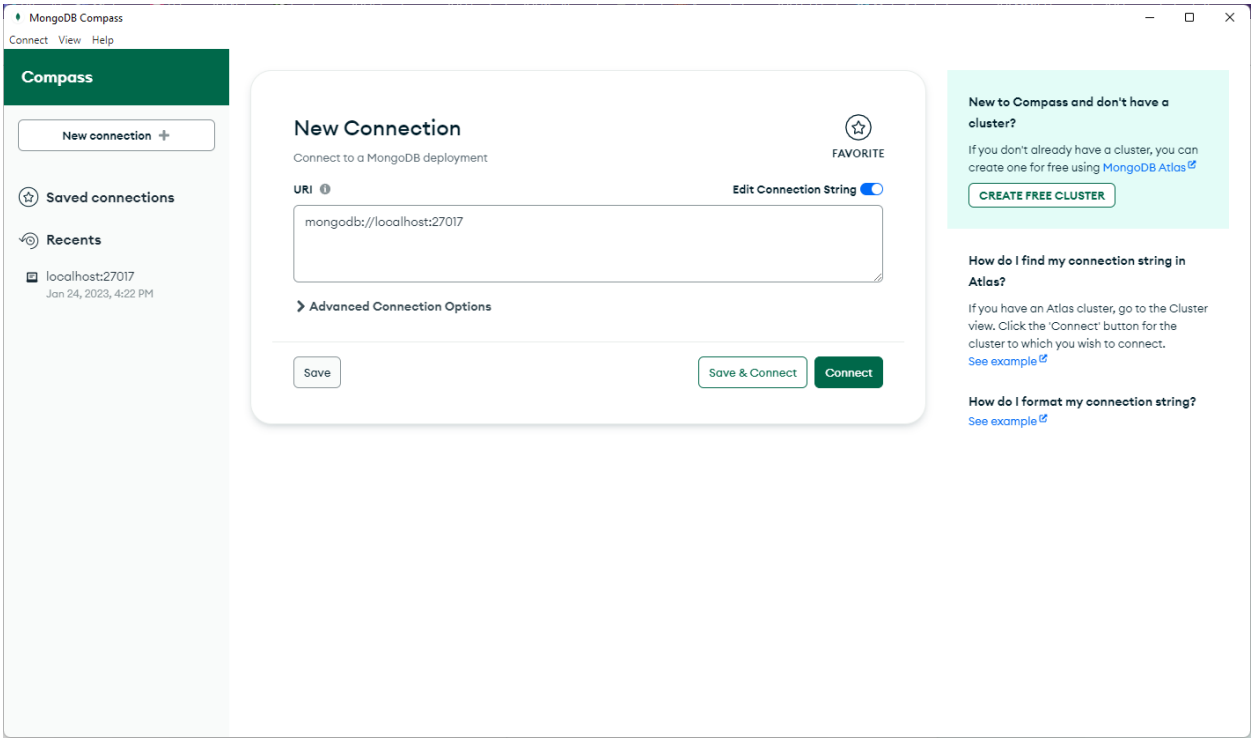
Ensure that **Install MongoDB Compass** is **checked**.





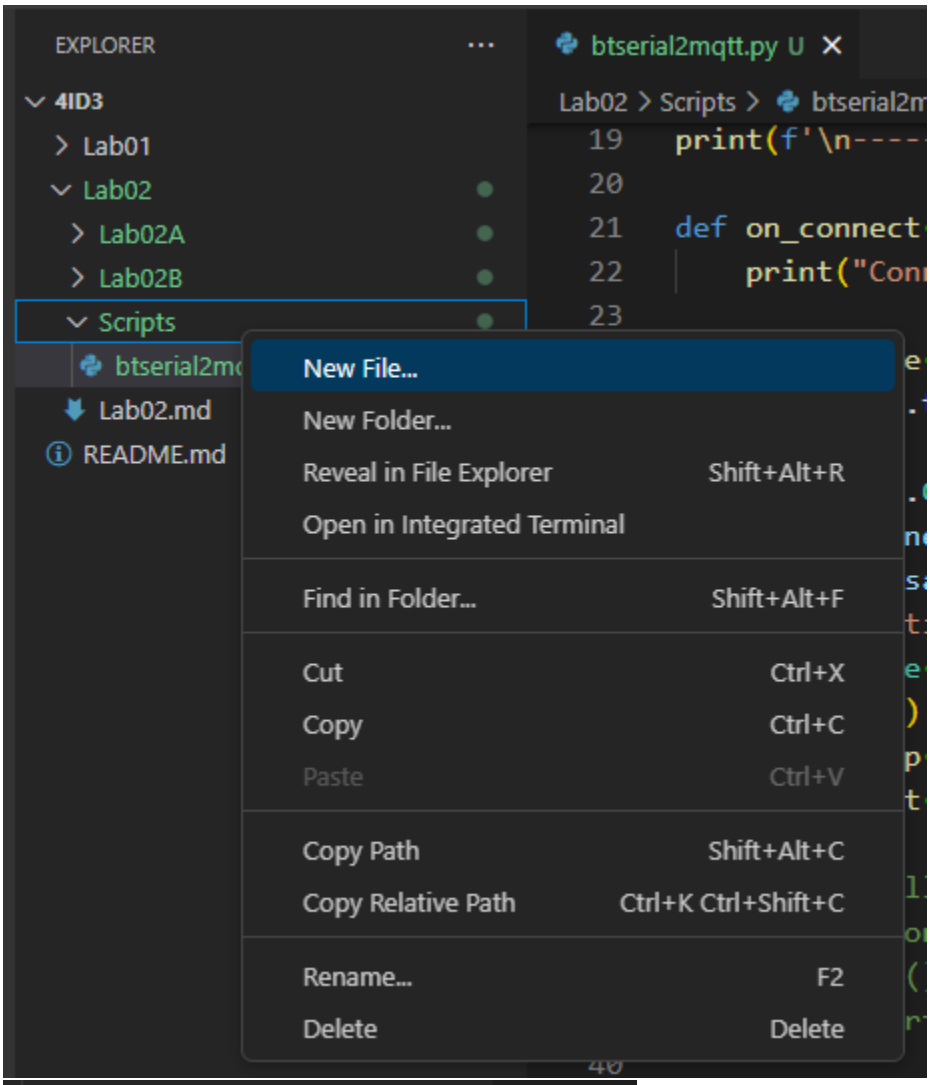
Launch **MongoDB Compass**.

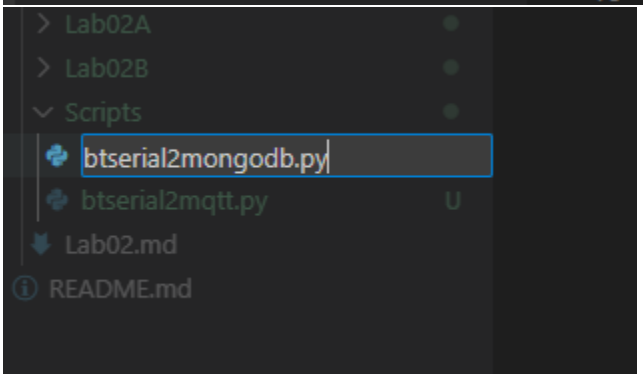




Press **Connect**.

Navigate back to **VSCode** and create a **new Python script** in the **Scripts/** folder named **btserial2mongodb**.

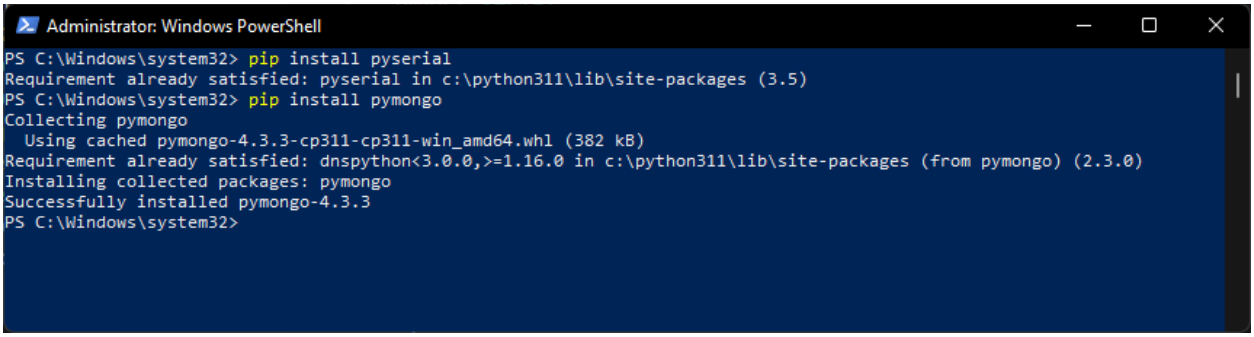




Using **Powershell** as **Administrator**, install the following dependencies using **pip package manager**.

pip install pyserial

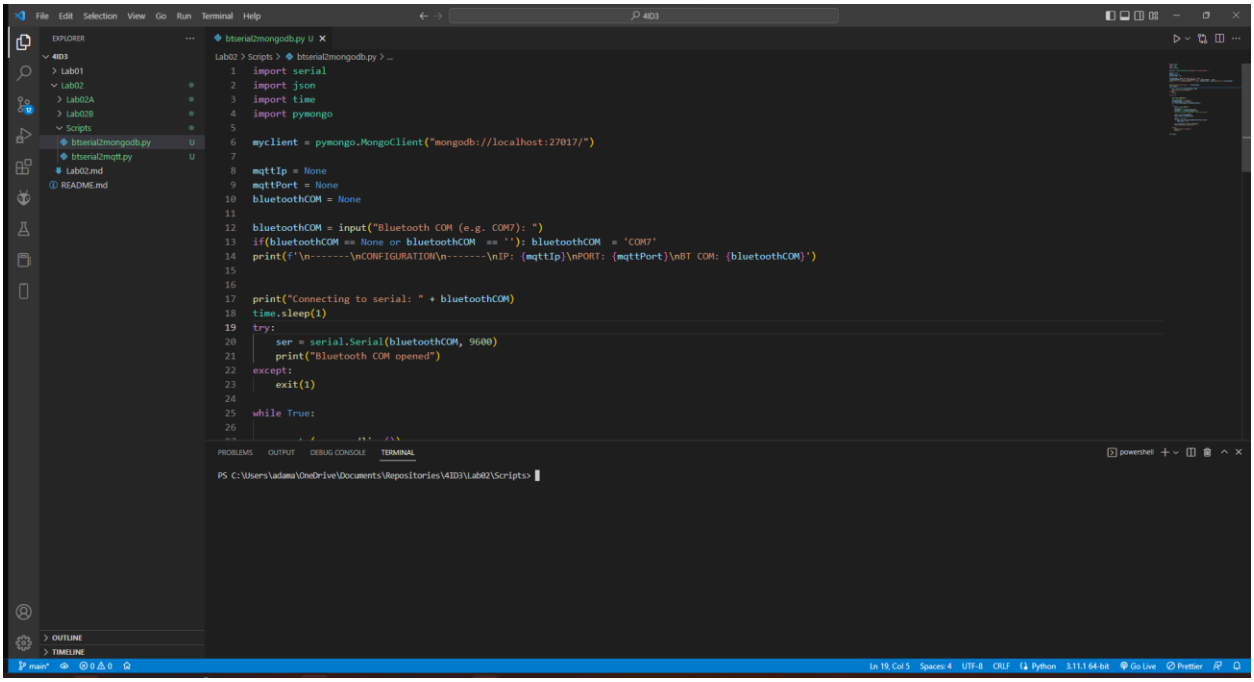
pip install pymongo



Paste the following Python script into the blank python file:

```
import serial
import json
import time
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")

bluetoothCOM = None


bluetoothCOM = input("Bluetooth COM (e.g. COM7): ")
if(bluetoothCOM == None or bluetoothCOM  == ''): bluetoothCOM  = 'COM7'
print(f'\n-------\nCONFIGURATION\n-------\nIP: {mqttIp}\nPORT: {mqttPort}\nBT COM: {bluetoothCOM}')


print("Connecting to serial: " + bluetoothCOM)
time.sleep(1)
try:
   ser = serial.Serial(bluetoothCOM, 9600)
   print("Bluetooth COM opened")
except:
   exit(1)

while True:

   cc=str(ser.readline())
   #cc = cc[6:][:-3]
   firstSplitIndex = cc.find('{')
   secondSplitIndex = cc.rfind('}')
   cc = cc[firstSplitIndex: secondSplitIndex+1]

   try:
      jDict = json.loads(cc)
      #print(jDict)
      groupName = list(jDict.keys())[0]
      deviceId = list(jDict[groupName])[0]
      #print(f'DeviceID: {deviceId} -> {jDict[deviceId]}')

      mydb = myclient[groupName]
      mycollection = mydb[deviceId]

      dbDict = dict({})
      for key, val in jDict[groupName][deviceId].items():
```

```
        dbDict[key] = val

    ret = mycollection.insert_one(dbDict)
    print("Inserted successfully")

except:
    print("Failed to decode: ")
    print(cc)


ser.close()
```



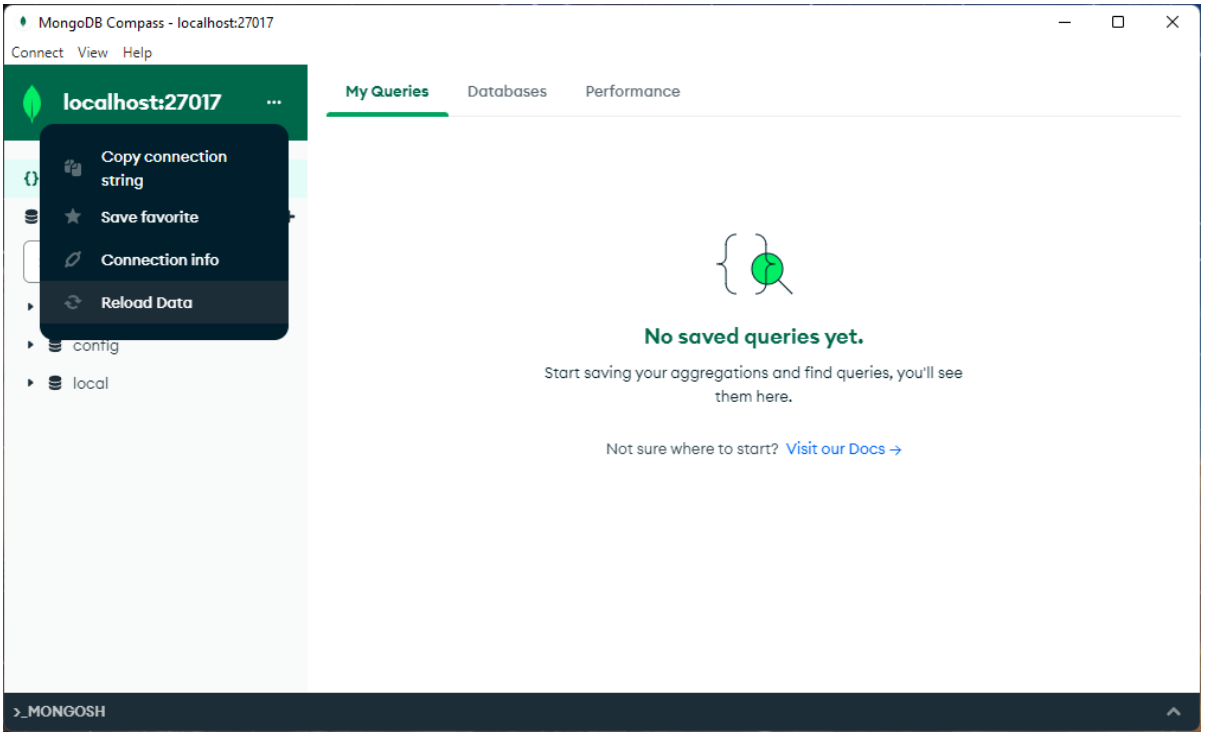
Run the python program while the MacIoT board is connected with Bluetooth and transmitting data.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3> cd .\Lab02\Scripts\
PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3\Lab02\Scripts> ls


    Directory: C:\Users\adama\OneDrive\Documents\Repositories\4ID3\Lab02\Scripts


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         2023-01-24   4:44 PM           1411 btserial2mongodb.py
-a----         2023-01-24   4:19 PM           2008 btserial2mqtt.py


PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3\Lab02\Scripts>
```

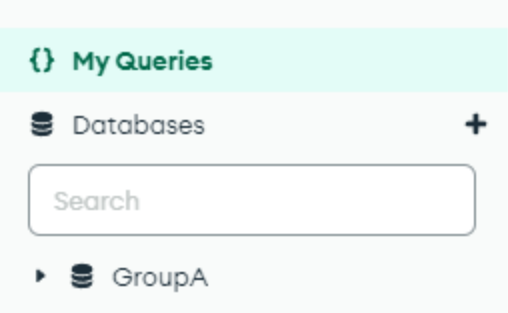
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3\Lab02\Scripts> python .\btserial2mongodb.py
Bluetooth COM (e.g. COM7): COM10

-------
CONFIGURATION
-------
IP: None
PORT: None
BT COM: COM10
Connecting to serial: COM10
Bluetooth COM opened
Inserted successfully
Inserted successfully
Inserted successfully
```

If the data is being parsed correctly and it can access the MongoDB database, then **Inserted Successfully** should be printed.

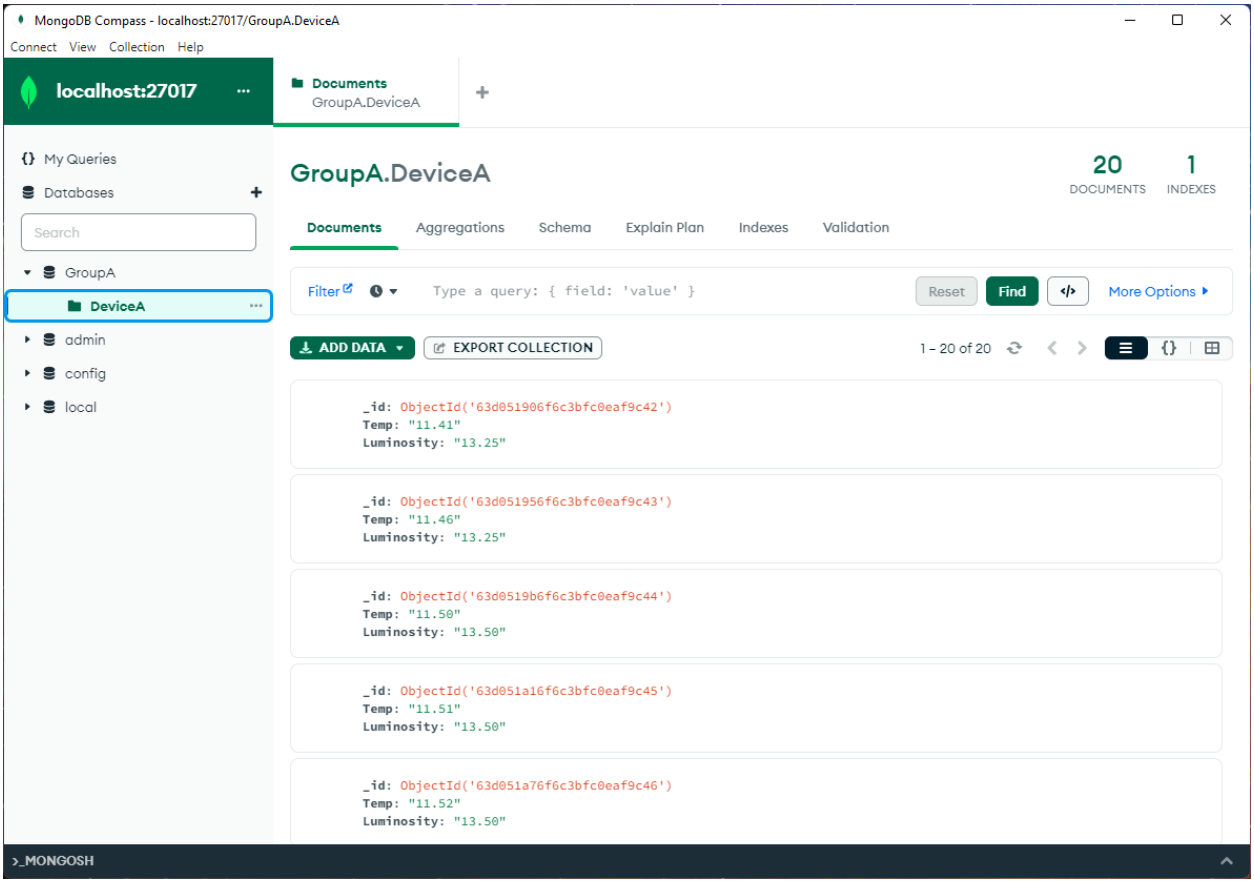
Reload your MongoDB Compass client:



Observe that a new database has been generated. Open this database.



You should see all of your data being inserted.

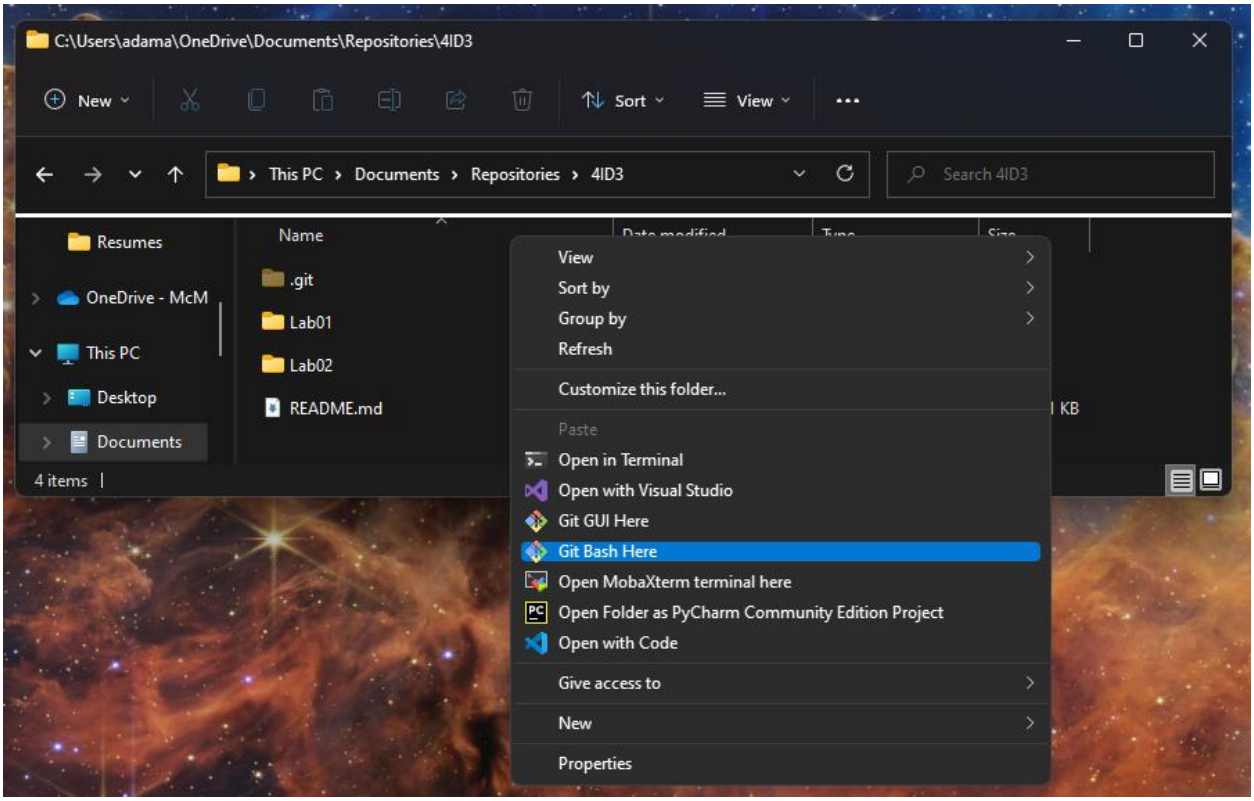


Optional: Exercise C

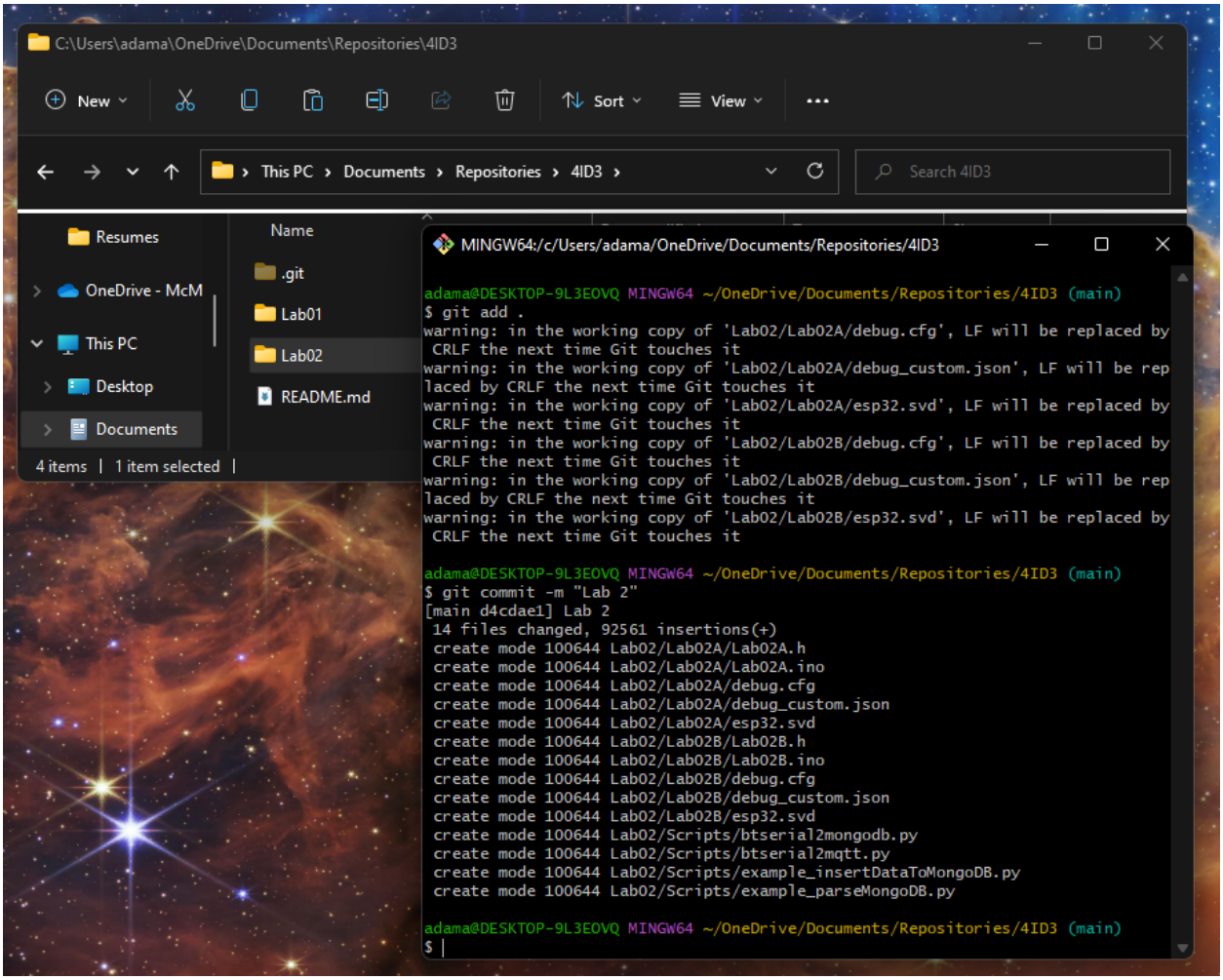
Collect sensor data for 2 minutes then export the dataset from the database. Using excel or another spreadsheet software, graph the data and identify temperature and light intensity trends. Include the graphs with your lab submission.

Pushing Changes to GitHub

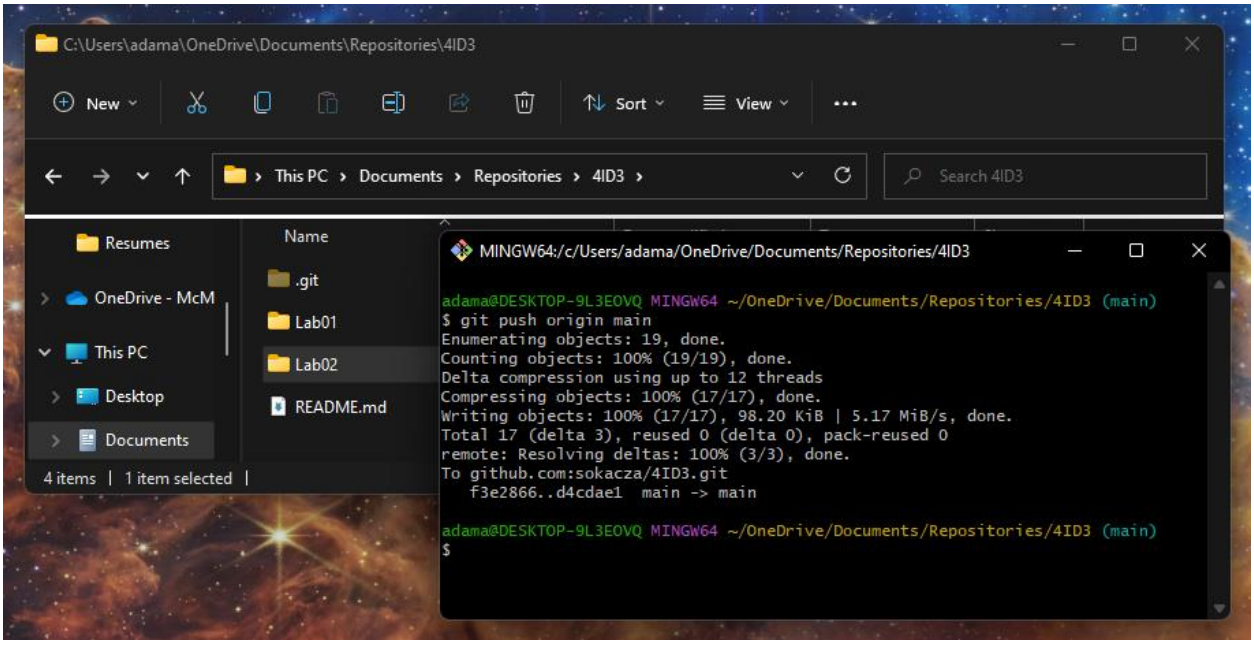
Open **git bash** in your local repository.



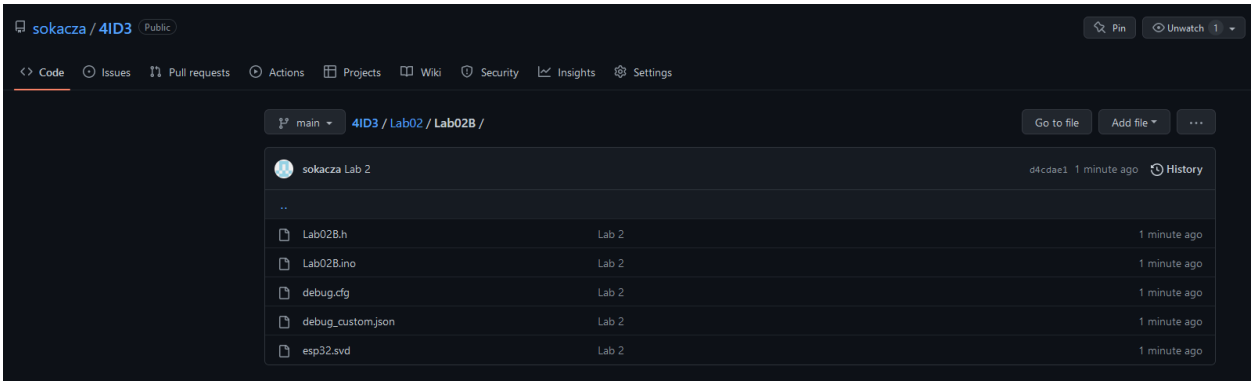
Add and commit your changes.



Push your changes.



Check that they've been synced correctly.



END