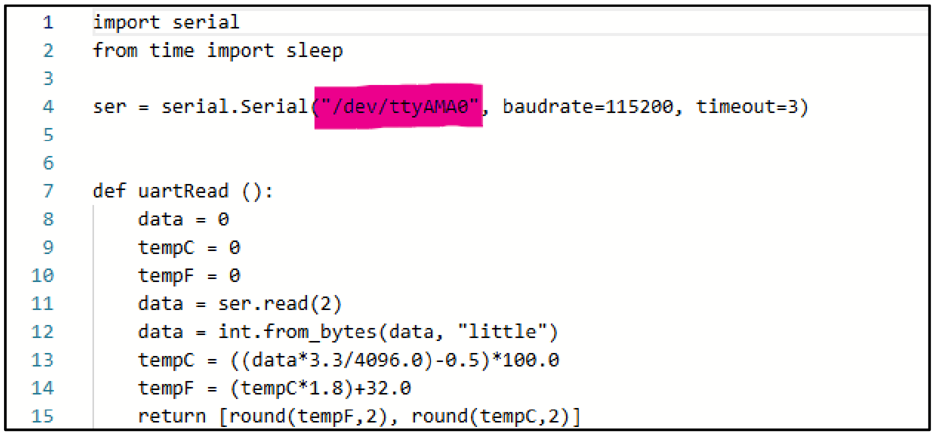
Raspberry Pi and Python for Communication, Data Logging, Data Analysis, and Visualization

To start this lab, we needed to set up and configure our Raspberry Pi 5 and STM32 Nucleo boards. First, we used an SD flasher to download the Raspberry Pi OS onto the SD card. After we inserted the SD, the Raspberry Pi was then ready to being code development. We then set up our STM board using CubeMX to configure a pin for ADC and a pin for UART transmission. We decided to replicate the same code we used in Lab 3 to record and transmit the temperature readings from the MCP9700A sensor. This meant that we were using a 12-bit ADC and transmitting 2 8-bit pieces for each temperature reading. We decided to continuously transmit readings from the STM board to ensure we would always have data to read when we ran our Raspberry Pi functions. On the Pi side we created a lab7.py file which would call all our other functions. In this file we created an array of size 20 for each of the different measurements we needed to collect as well as a dataCount variable to keep track of indexing. We then created a while loop that check for dataCount < 20.

```
MCP_F = [None]*20
MCP_C = [None]*20
DHT_F = [None]*20
DHT_C = [None]*20
DHT_H = [None]*20
ERR_F = [None]*20
ERR_C = [None]*20
```

lab7.py

Before we began working on receiving the data on the Pi side, we wanted to explore GPIO implementation on the Raspberry Pi. We created a simple python program that would alternate between turning GPIO23 pin high and then low. We then connected an LED to this output. Before we could use UART on our Raspberry Pi, we enabled UART in the terminal by using the command sudo raspi-config to access the Raspberry Pi's software configuration tool. In the configuration tool we disabled the serial login shell and enabled the serial interface. Next to receive the transmitted data, we created a lab7uart.py file. We found the UART mapping using the following command in the terminal: ls –l /dev. Within the lab7uart.py file we created a variable called ser which is an instance of serial.Serial using the mapping value for the terminal and the baud rate from CubeMX. This variable allows us to interact with serial communication ports like the UART Rx pin on GPIO15 of the Raspberry Pi. We were then able to use the read function to read 2 bytes of data, and the int.from_bytes function to convert the two bytes of little-endian data into its int representation. Finally, just like in Lab 3, we converted the measurement into Celsius and Fahrenheit units. Initially, serial communication was not working as expected. To troubleshoot this, we made sure to check CubeMX to verify the baud rate and used the Keil debugger to verify that the issue was not on the STM board side of the communication. Additionally, we added a timeout of 3 seconds to prevent the code from running indefinitely if no data is being received. After consulting with another team, we discovered that we had misinterpreted the UART mapping in the terminal and were using the incorrect value for the serial port. The correct value is highlighted below. After we made this correction, the serial communication functioned properly, and we were able to call the UART function in the while loop of our main python code.



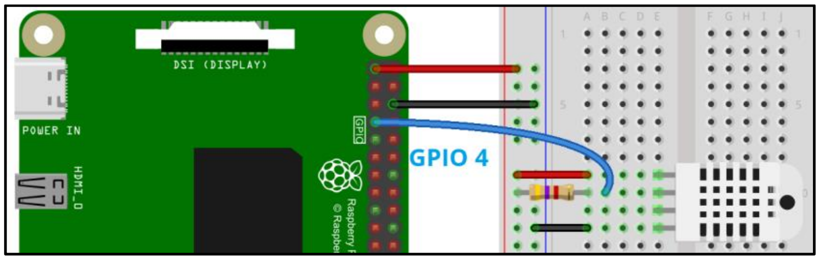
```python
import serial
from time import sleep

ser = serial.Serial("/dev/ttyAMA0", baudrate=115200, timeout=3)


def uartRead ():
    data = 0
    tempC = 0
    tempF = 0
    data = ser.read(2)
    data = int.from_bytes(data, "little")
    tempC = ((data*3.3/4096.0)-0.5)*100.0
    tempF = (tempC*1.8)+32.0
    return [round(tempF,2), round(tempC,2)]
```

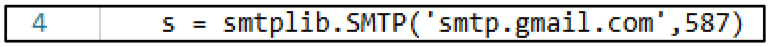
lab7uart.py UART Raspberry Pi Code

Next, we worked on getting temperature and humidity readings from the DHT11 sensor. Based on our research on this sensor we didn't need to do the initialization and reading manually we just needed to import the adafruit_dht module. In a separate lab7dht.py file we created an adafruit.DHT11 instance at GPIO4, a function to read the sensor, and a function to exit the sensor all based on a DHT11 implementation tutorial. When we ran the code, we began getting errors in the terminal about "board" not being recognized. To fix this, we tried importing the board module, but this did not resolve the error. After reading up on Adafruit we found out there is a board module already built into the module. Importing the CircuitPython board modules overrides the included module and causes more issues. After encountering numerous problems, we decided to reboot the Raspberry Pi. However, we could not access the reboot menu, so we attempted to reboot using the command line prompt which caused our Raspberry Pi to never turn back on. We tried to recover our files off the Raspberry Pi using the SD flasher but were unable to get any of our files back. To make our Pi operational again we needed a new OS, so we wiped the SD card and repeated our previous steps. Finally, we discovered the issue with our DHT11 sensor configuration was not in the software, but that our sensor was not getting enough voltage. We moved the Vin pin from the 3.3V source to the 5V source and it began to work properly.



DHT11 wiring for Raspberry Pi 4 which we adapted for Raspberry Pi 5

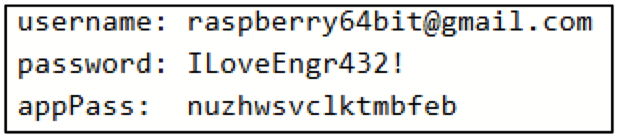
With both the sensors working properly within our while loop, we calculated the percept error for the different measurements. To add email alerts, we followed the instructions from an online tutorial. We used the SMTP protocol and Python library to set up email alerts. First, we needed to create a connection specifying the sever location and port as shown below.



```
    s = smtplib.SMTP('smtp.gmail.com',587)
```

Lab7email.py

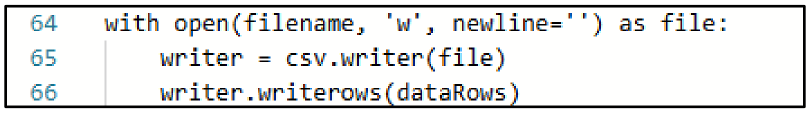
For security purposes we created a new Gmail address for this project and attempted to login. We were unsuccessful and discovered that the protections used to login to a google account would not allow us to directly login using the login command with our actual password. After researching the error, we discovered that it could be fixed by turning off login protections in our Gmail account, but we were unable to do this because our account was too new and that setting was being deprecated. This was a 16-character password that would be used specifically for our Raspberry Pi logins. To implement this functionality into our lab, we added an if statement in our while loop that checked if temperatures were above 20 degrees Celsius and then called our email function if an alert was necessary.



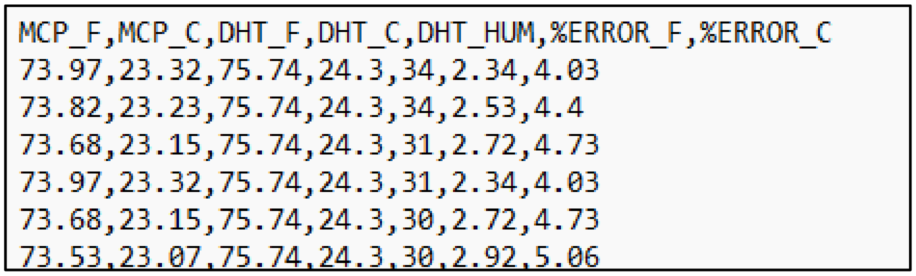
```
username: raspberry64bit@gmail.com
password: ILoveEngr432!
appPass:  nuzhwsvclktmbfeb
```

Our new Gmail address with our actual password and our app password

To be able to write our data to a csv file we used a 2D array called dataRows. With each iteration of the while loop we stored the data in a 1D array for each type of measurement and in a column of a 2D array. The 1D array would be used for later analysis and visualization, and the 2D array would be used to write to the csv file. We imported the csv module and then used the following three lines to write the data to a file with the name stored in filename, in this case we used the name lab7tempData.csv.



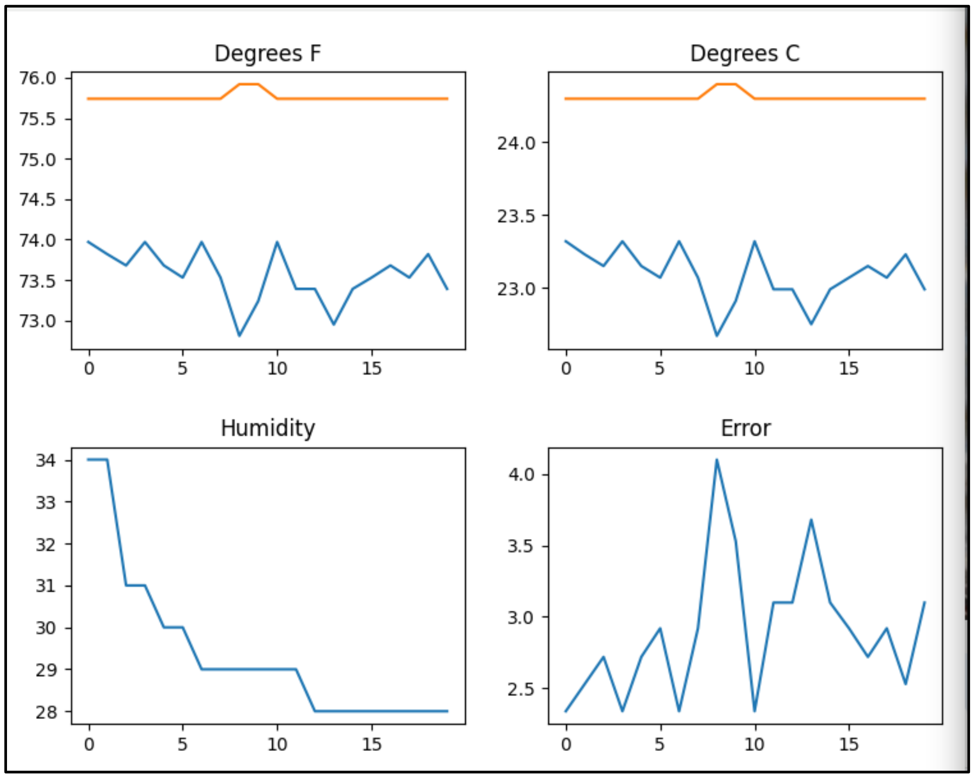
```
with open(filename, 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(dataRows)
```



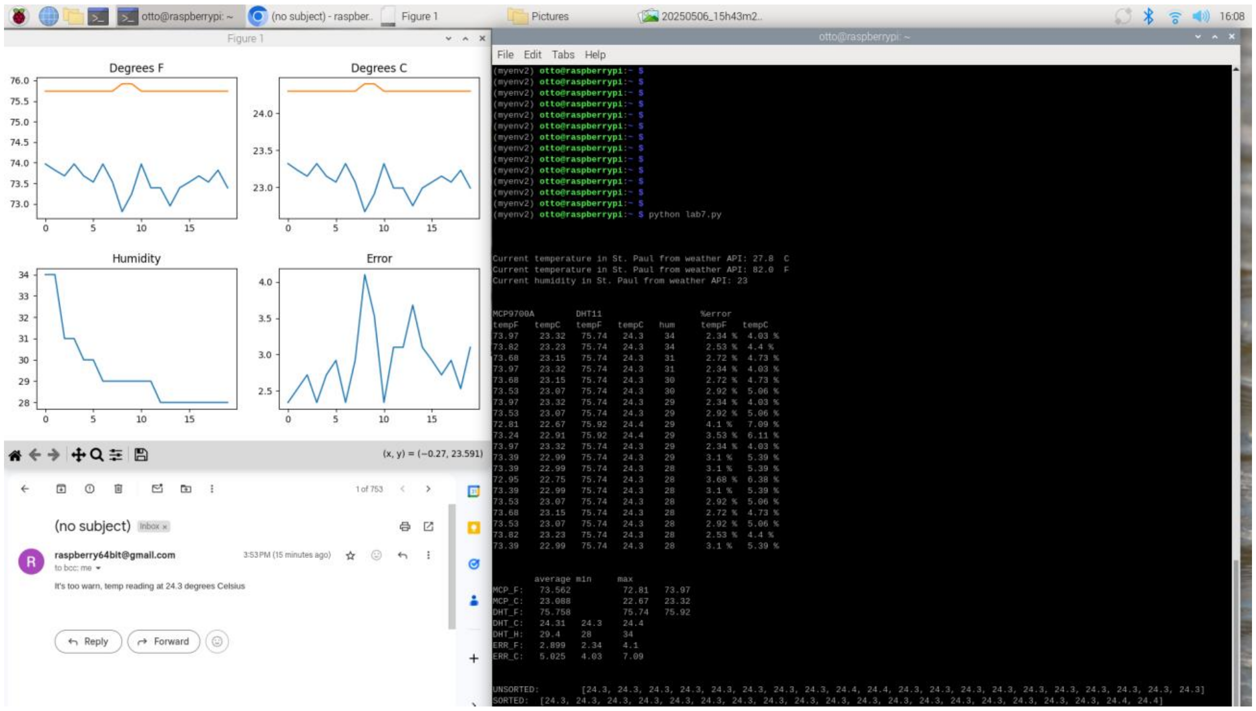
```
MCP_F,MCP_C,DHT_F,DHT_C,DHT_HUM,%ERROR_F,%ERROR_C
73.97,23.32,75.74,24.3,34,2.34,4.03
73.82,23.23,75.74,24.3,34,2.53,4.4
73.68,23.15,75.74,24.3,31,2.72,4.73
73.97,23.32,75.74,24.3,31,2.34,4.03
73.68,23.15,75.74,24.3,30,2.72,4.73
73.53,23.07,75.74,24.3,30,2.92,5.06
```

The first 6 data points as they are stored in the csv file lab7TempData.csv

At the end of the loop, we exited the DHT11 sensor and began working on our analysis and visualization. We created a file called lab7visualize.py which contained a function to calculate the average, min, and max value from an array of numbers. We then used these functions to print the corresponding values for each measurement. After brushing up on the quicksort method, in another separate file called lab7quicksort.py we wrote our own quicksort function. Finally, we used the matplotlib library to plot graphs of the temperatures in Celsius and Fahrenheit, the humidity, and the error.



To implement one additional library, we decided to import the requests and JSON libraries and then connect to a weather API. To allow us to make API calls we need to find a free weather API that would allow us to make an account and generate our own API key. Once we were able to successfully connect to the API, we needed to print the response data and determine which values we wanted to display. We did this by taking the response data and turning it into a JSON file and then using the value labels as array indexes to extract the values. This process was relatively simple and allows us to print the temperature in Celsius, temperature in Fahrenheit, and the humidity of St. Paul which gave context to the accuracy of our readings.



(no subject)

raspberry64bit@gmail.com

to bcc: me

3:53 PM (15 minutes ago)

It's too warn, temp reading at 24.3 degrees Celsius

```
(myenv2) otto@raspberrypi:~ $
(myenv2) otto@raspberrypi:~ $
(myenv2) otto@raspberrypi:~ $
(myenv2) otto@raspberrypi:~ $
(myenv2) otto@raspberrypi:~ $
(myenv2) otto@raspberrypi:~ $
(myenv2) otto@raspberrypi:~ $
(myenv2) otto@raspberrypi:~ $
(myenv2) otto@raspberrypi:~ $
(myenv2) otto@raspberrypi:~ $
(myenv2) otto@raspberrypi:~ $
(myenv2) otto@raspberrypi:~ $
(myenv2) otto@raspberrypi:~ $
(myenv2) otto@raspberrypi:~ $ python lab7.py



Current temperature in St. Paul from weather API: 27.8  C
Current temperature in St. Paul from weather API: 82.0  F
Current humidity in St. Paul from weather API: 23


MCP9700A        DHT11                   %error
tempF   tempC   tempF   tempC   hum      tempF   tempC
73.97    23.32   75.74   24.3    34       2.34 %  4.03 %
73.82    23.23   75.74   24.3    34       2.53 %  4.4 %
73.68    23.15   75.74   24.3    31       2.72 %  4.73 %
73.97    23.32   75.74   24.3    31       2.34 %  4.03 %
73.68    23.15   75.74   24.3    30       2.72 %  4.73 %
73.53    23.07   75.74   24.3    30       2.92 %  5.06 %
73.97    23.32   75.74   24.3    29       2.34 %  4.03 %
73.53    23.07   75.74   24.3    29       2.92 %  5.06 %
72.81    22.67   75.92   24.4    29       4.1 %   7.09 %
73.24    22.91   75.92   24.4    29       3.53 %  6.11 %
73.97    23.32   75.74   24.3    29       2.34 %  4.03 %
73.39    22.99   75.74   24.3    29       3.1 %   5.39 %
73.39    22.99   75.74   24.3    28       3.1 %   5.39 %
72.95    22.75   75.74   24.3    28       3.68 %  6.38 %
73.39    22.99   75.74   24.3    28       3.1 %   5.39 %
73.53    23.07   75.74   24.3    28       2.92 %  5.06 %
73.68    23.15   75.74   24.3    28       2.72 %  4.73 %
73.53    23.07   75.74   24.3    28       2.92 %  5.06 %
73.82    23.23   75.74   24.3    28       2.53 %  4.4 %
73.39    22.99   75.74   24.3    28       3.1 %   5.39 %



        average min     max
MCP_F:   73.562          72.81   73.97
MCP_C:   23.088          22.67   23.32
DHT_F:   75.758          75.74   75.92
DHT_C:   24.31   24.3    24.4
DHT_H:   29.4    28      34
ERR_F:   2.899   2.34    4.1
ERR_C:   5.025   4.03    7.09


UNSORTED:        [24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.4, 24.4, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3]
SORTED:  [24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.3, 24.4, 24.4]
```

Lab 7 Results

<u>Sources</u>

UART: https://www.electronicwings.com/raspberry-pi/raspberry-pi-uart-communication-using-python-and-c

DHT11: https://randomnerdtutorials.com/raspberry-pi-dht11-dht22-python/

Email: https://www.geeksforgeeks.org/send-mail-gmail-account-using-python/