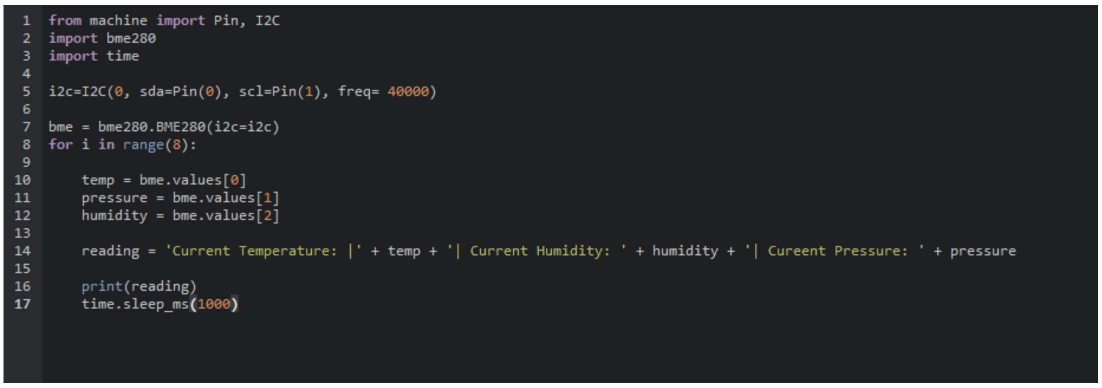
Hamza Lasry – w1804929
Internet of Things – 6NTCM009W.2

1. DEVELOPMENT:

A) This piece of code is designed to read temperatures such as temperature, humidity and pressure throughout the use of the BME280 sensor. The first thing to do is to import all the necessary libraries for interacting with the hardware, bme280 for the sensor, I2C for the communication and the machine for the GPIO pin, following next we have to set up the I2C communication channel by defining the pins used for data, the clock(SCL) and the frequency. In line number 7, we have initialized the BME in order to start communicating, and we enter a loop that will be executed eight times, which will read the temperature, pressure and humidity.

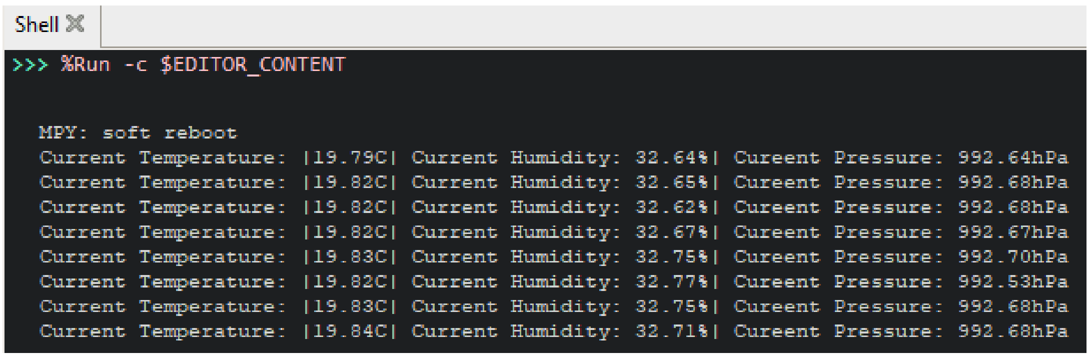
Input



```
1   from machine import Pin, I2C
2   import bme280
3   import time
4
5   i2c=I2C(0, sda=Pin(0), scl=Pin(1), freq= 40000)
6
7   bme = bme280.BME280(i2c=i2c)
8   for i in range(8):
9
10      temp = bme.values[0]
11      pressure = bme.values[1]
12      humidity = bme.values[2]
13
14      reading = 'Current Temperature: |' + temp + '| Current Humidity: ' + humidity + '| Cureent Pressure: ' + pressure
15
16      print(reading)
17      time.sleep_ms(1000)
```

[Fig.1 Input 1st task]

Output

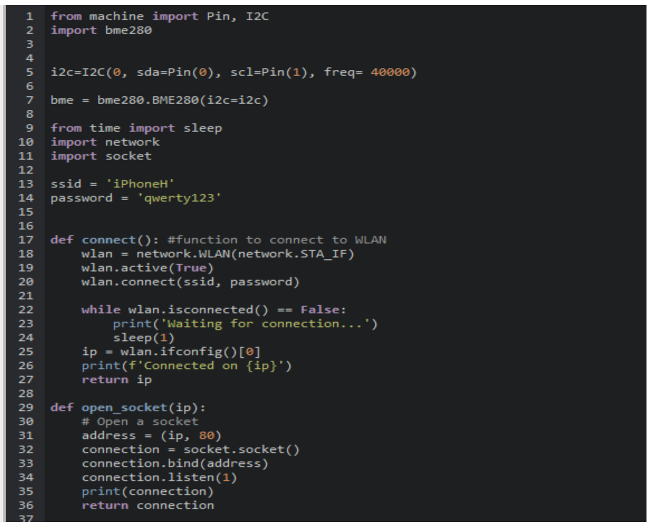


```
>>> %Run -c $EDITOR_CONTENT

  MPY: soft reboot
  Current Temperature: |19.79C| Current Humidity: 32.64%| Cureent Pressure: 992.64hPa
  Current Temperature: |19.82C| Current Humidity: 32.65%| Cureent Pressure: 992.68hPa
  Current Temperature: |19.82C| Current Humidity: 32.62%| Cureent Pressure: 992.68hPa
  Current Temperature: |19.82C| Current Humidity: 32.67%| Cureent Pressure: 992.67hPa
  Current Temperature: |19.83C| Current Humidity: 32.75%| Cureent Pressure: 992.70hPa
  Current Temperature: |19.82C| Current Humidity: 32.77%| Cureent Pressure: 992.53hPa
  Current Temperature: |19.83C| Current Humidity: 32.75%| Cureent Pressure: 992.68hPa
  Current Temperature: |19.84C| Current Humidity: 32.71%| Cureent Pressure: 992.68hPa
```

[Fig.2 Output 1st task]

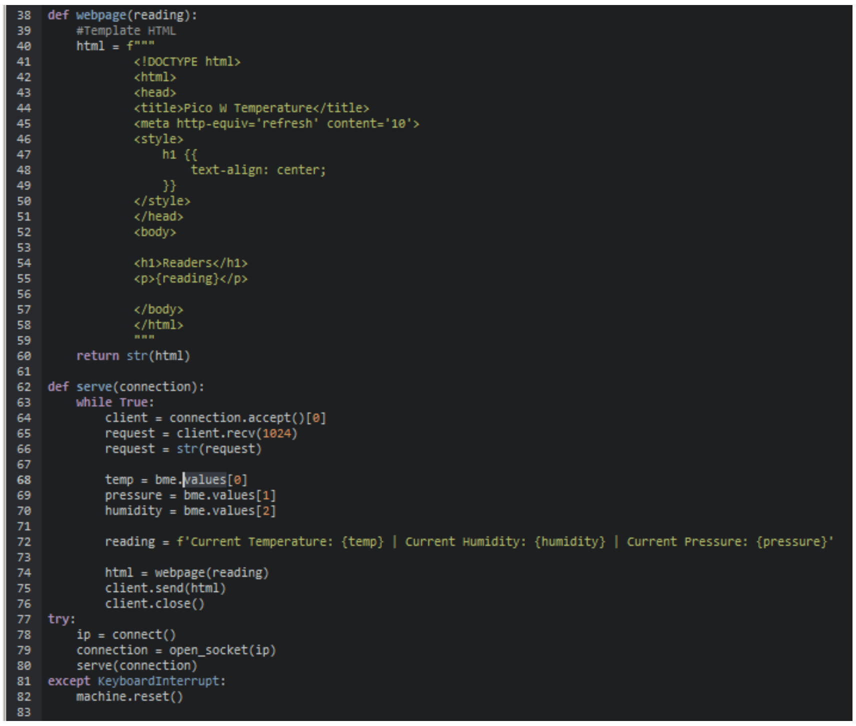
B) This script connects a microcontroller to a Wi-Fi network and sets up a server that reads data from a BME280 sensor. It serves a webpage that displays real-time temperature, humidity, and pressure readings. The microcontroller initializes an I2C connection to communicate with the sensor, connects to the specified Wi-Fi network, and listens for incoming HTTP requests. When a client connects by using the IP address printed, it sends an HTML page containing the latest sensor readings, which auto-refreshes every 10 seconds.

Input



```python
from machine import Pin, I2C
import bme280


i2c=I2C(0, sda=Pin(0), scl=Pin(1), freq= 40000)

bme = bme280.BME280(i2c=i2c)

from time import sleep
import network
import socket

ssid = 'iPhoneH'
password = 'qwerty123'


def connect(): #function to connect to WLAN
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, password)

    while wlan.isconnected() == False:
        print('Waiting for connection...')
        sleep(1)
    ip = wlan.ifconfig()[0]
    print(f'Connected on {ip}')
    return ip

def open_socket(ip):
    # Open a socket
    address = (ip, 80)
    connection = socket.socket()
    connection.bind(address)
    connection.listen(1)
    print(connection)
    return connection
```

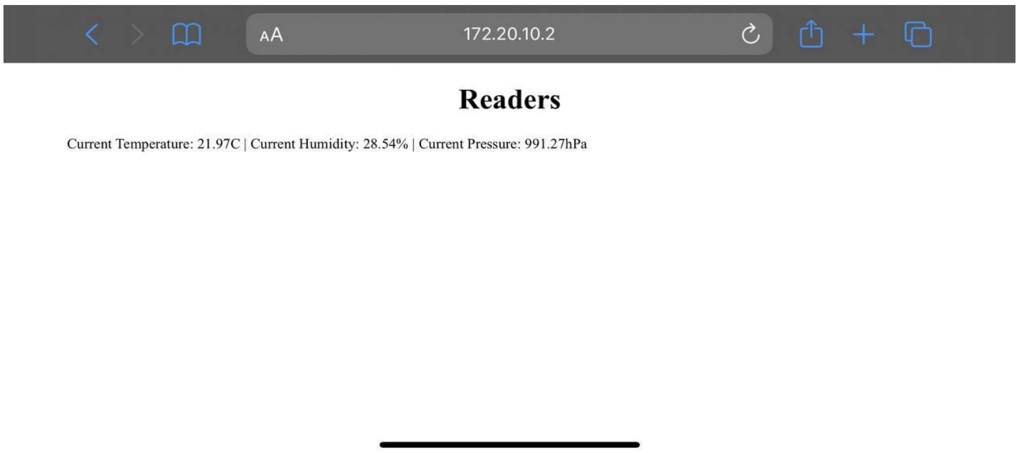
[Fig.3 Input 2nd task]



```python
def webpage(reading):
    #Template HTML
    html = f"""
            <!DOCTYPE html>
            <html>
            <head>
            <title>Pico W Temperature</title>
            <meta http-equiv='refresh' content='10'>
            <style>
                h1 {{
                    text-align: center;
                }}
            </style>
            </head>
            <body>

            <h1>Readers</h1>
            <p>{reading}</p>

            </body>
            </html>
            """
    return str(html)

def serve(connection):
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        request = str(request)

        temp = bme.values[0]
        pressure = bme.values[1]
        humidity = bme.values[2]

        reading = f'Current Temperature: {temp} | Current Humidity: {humidity} | Current Pressure: {pressure}'

        html = webpage(reading)
        client.send(html)
        client.close()
try:
    ip = connect()
    connection = open_socket(ip)
    serve(connection)
except KeyboardInterrupt:
    machine.reset()
```

[Fig.4 Input 2nd task]

Output



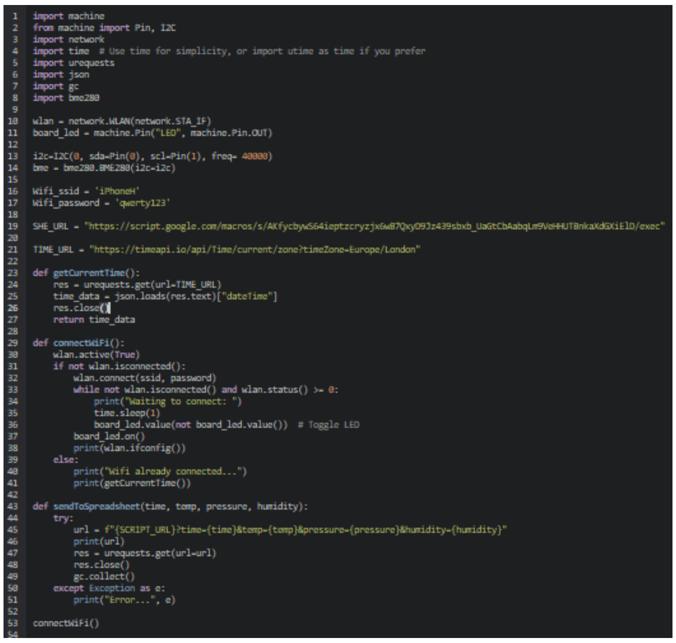
Readers

Current Temperature: 21.97C | Current Humidity: 28.54% | Current Pressure: 991.27hPa

[Fig.5 Output 2nd task]

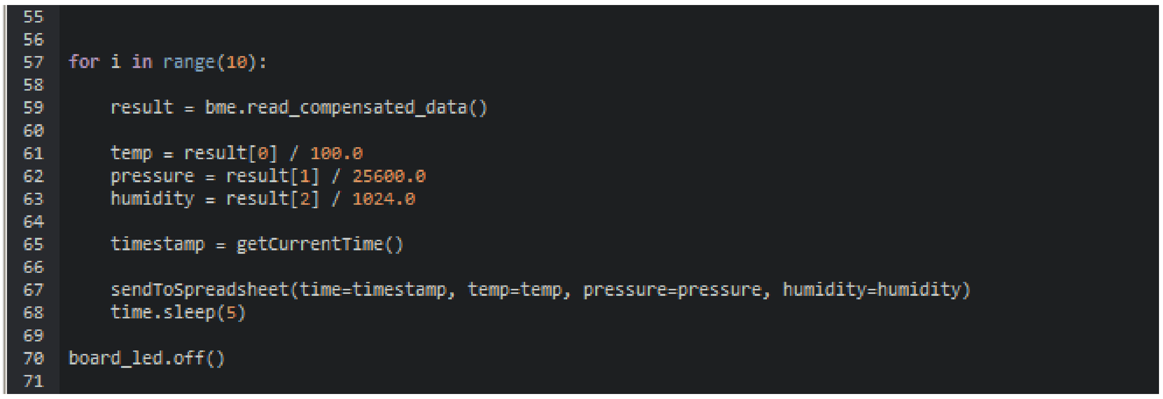
C) The Python script is designed to function as a serverless cloud application that automatically logs sensor data into a Google Sheet. Upon execution, the script reads environmental measurements from a sensor and uses Google Apps Script to process and record this data remotely into a spreadsheet. This setup not only facilitates the seamless storage of data but also allows for real-time monitoring and analysis directly from Google Sheets. Through the use of Apps Script, the system handles, processes, and stores the sensor data efficiently, making it accessible from anywhere with internet connectivity.

Input



```python
import machine
from machine import Pin, I2C
import network
import time  # Use time for simplicity, or import utime as time if you prefer
import urequests
import json
import gc
import bme280

wlan = network.WLAN(network.STA_IF)
board_led = machine.Pin("LED", machine.Pin.OUT)

i2c=I2C(0, sda=Pin(0), scl=Pin(1), freq= 40000)
bme = bme280.BME280(i2c=i2c)

Wifi_ssid = 'iPhoneH'
Wifi_password = 'qwerty123'

SHE_URL = "https://script.google.com/macros/s/AKfycbywS64ieptzcryzjx6w87Qxy09Jz439sbxb_UaGtCbAabqLm9VeHHUTBnkaXdGXiE1D/exec"

TIME_URL = "https://timeapi.io/api/Time/current/zone?timeZone=Europe/London"

def getCurrentTime():
    res = urequests.get(url=TIME_URL)
    time_data = json.loads(res.text)["dateTime"]
    res.close()
    return time_data

def connectWiFi():
    wlan.active(True)
    if not wlan.isconnected():
        wlan.connect(ssid, password)
        while not wlan.isconnected() and wlan.status() >= 0:
            print("Waiting to connect: ")
            time.sleep(1)
            board_led.value(not board_led.value())  # Toggle LED
        board_led.on()
        print(wlan.ifconfig())
    else:
        print("Wifi already connected...")
        print(getCurrentTime())

def sendToSpreadsheet(time, temp, pressure, humidity):
    try:
        url = f"{SCRIPT_URL}?time={time}&temp={temp}&pressure={pressure}&humidity={humidity}"
        print(url)
        res = urequests.get(url=url)
        res.close()
        gc.collect()
    except Exception as e:
        print("Error...", e)

connectWiFi()
```

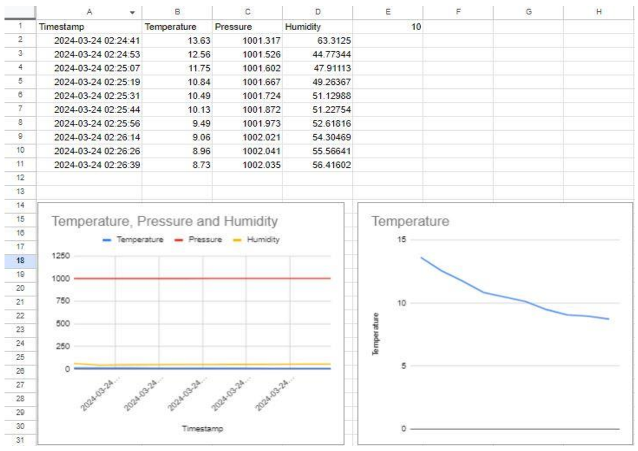
[Fig.6 Input 3rd task]



```


for i in range(10):

    result = bme.read_compensated_data()

    temp = result[0] / 100.0
    pressure = result[1] / 25600.0
    humidity = result[2] / 1024.0

    timestamp = getCurrentTime()

    sendToSpreadsheet(time=timestamp, temp=temp, pressure=pressure, humidity=humidity)
    time.sleep(5)

board_led.off()

```

[Fig.7 Input 3rd task]

Output



Timestamp	Temperature	Pressure	Humidity	10
2024-03-24 02:24:41	13.63	1001.317	63.3125	
2024-03-24 02:24:53	12.56	1001.526	44.77344	
2024-03-24 02:25:07	11.75	1001.602	47.91113	
2024-03-24 02:25:19	10.84	1001.667	49.26367	
2024-03-24 02:25:31	10.49	1001.724	51.12988	
2024-03-24 02:25:44	10.13	1001.872	51.22754	
2024-03-24 02:25:56	9.49	1001.973	52.61816	
2024-03-24 02:26:14	9.06	1002.021	54.30469	
2024-03-24 02:26:26	8.96	1002.041	55.56641	
2024-03-24 02:26:39	8.73	1002.035	56.41602	

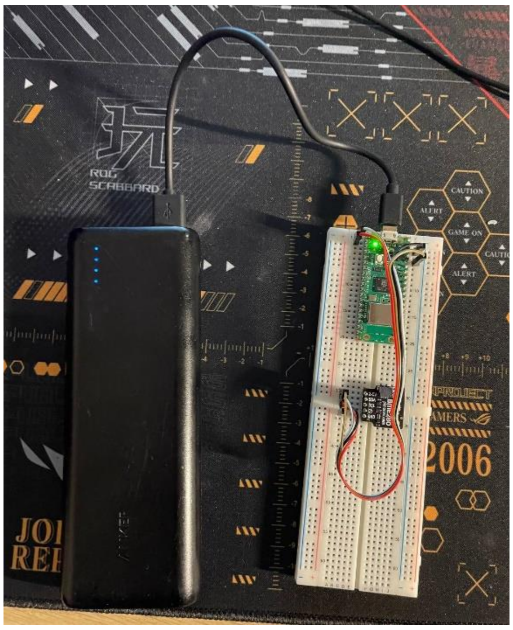
[Fig. 8 Output 3rd task]

2. TESTING :

During the testing phase, we established the Raspberry Pico by connecting it to a power bank. This procedure facilitated the acquisition of data such as temperature, pressure, and humidity. Notably, the Raspberry Pico operated independently from a laptop as the required code was already stored on the board. This methodology allowed for the collection of data in an autonomous and efficient manner.

Below we have attached the images of the setup and the results produced by the board, the results were performed in three different locations: inside a fridge, indoor and outdoor. Once done, we plotted the temperature of the different locations in a graph for a better understanding.

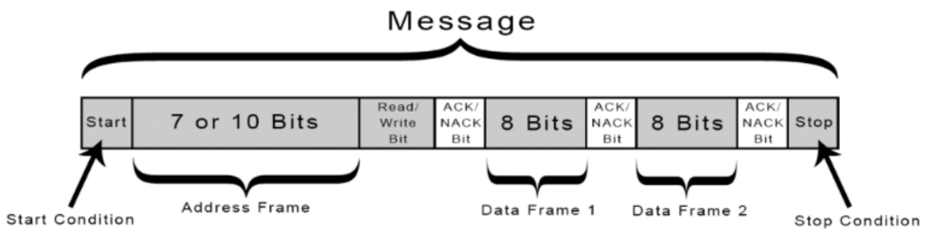
In Figure 15, we have compiled a comprehensive list of all relevant temperature readings to our scenario. To ensure accuracy, we have taken 10 readings for each environment. The resulting line graph effectively illustrates the changes in temperature across the different environments, with each environment clearly labelled for ease of reference.


[Fig.9 Raspberry Pico W Setup]

3. Report
1. a description on the overall functionality of the prototype setup

The prototype setup commenced with the establishment of our hardware infrastructure, employing the Raspberry Pi Pico W, a microcontroller development board renowned for its suitability in crafting diverse IoT projects, characterized by minimal power consumption and integrated Wi-Fi connectivity. The board is responsible for executing the Python scripts written to complete the tasks required.

BME280 sensor, which is installed into our board and provided by the university is a sensor made by the famous German manufacturer Bosch and is able to measure pressure, temperature and humidity, the transmission of data/messages is made using I2C, a serial communication protocol, during this process each message is broken up into frames of data, every single message has an address frame that contains a binary address and contains different condition such as a start and stop, read and write bits and ACK/NACK bits.



[Fig.10 I2C Protocol Message Frame Structure]

Connection is established by using the Raspberry board, integrated with wi-fi and external networks, in our case we have used a mobile hotspot to allow the activation of access and transfer data from a distance, this involves setting up the Wi-Fi credentials using SSID and password of the hotspot, once done the board will find and connect to the network.

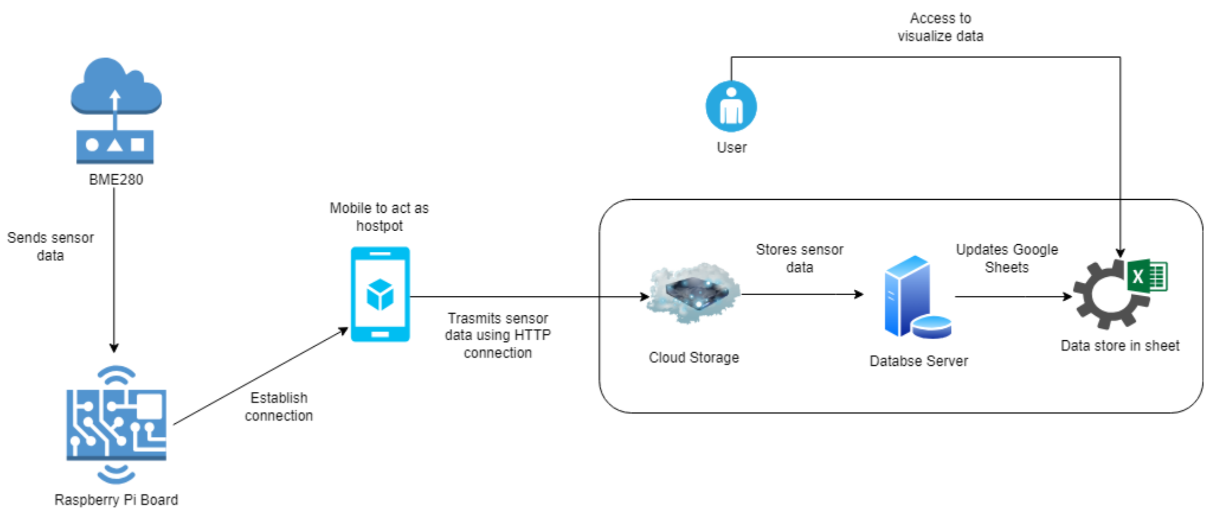
To achieve our third objective of transmitting and storing data to a cloud app, we've utilized the "App Scripts" extension within Google Sheets. This involves selecting the specific data to be transmitted and customizing the script accordingly. Following the customization, the script is deployed, which generates a URL. This URL is then integrated into our Python script, enabling the sending of data across the network to the web app.

3.An explanation of why this setup represents an IoT system

“The Internet of Things refers to a network of physical devices, vehicles, appliances, and other physical objects that are embedded with sensors, software and network connectivity, allowing them to collect and share data”, this definition stated by IBM emphasises that IoT systems collect data, enabling them to monitor, control and analyse various aspects of the physical world.

In our scenario, we are utilizing Wi-Fi connectivity to access online resources such as time API and Google Sheets, and the board integrates a sensor(bme280) that is able to gather real-world environment data. After being collected, the data is sent over an HTTP protocol, making the communication seamless with the other app/platforms online. The processing unit and power supply represent the Raspberry Pi Pico W and the power bank, as shown in our setup in Figure 9. Sensing, processing, connectivity and data processing are the backbone of the Internet of Things.

Another pillar of an IoT system is remote access to data, which enables users to monitor and analyse data from anywhere with internet access. To create an Internet of Things device, it's essential to have all the necessary components in place, each component mentioned plays a vital role in making the device functional and efficient and considered as an IoT device. Furthermore, it is essential to have a robust communication protocol when dealing with data, especially sensitive/important ones as some data needs to be received in time and accurately. In the diagram below I have visualised how the different components of this system interact with each other to form a cohesive IoT system:

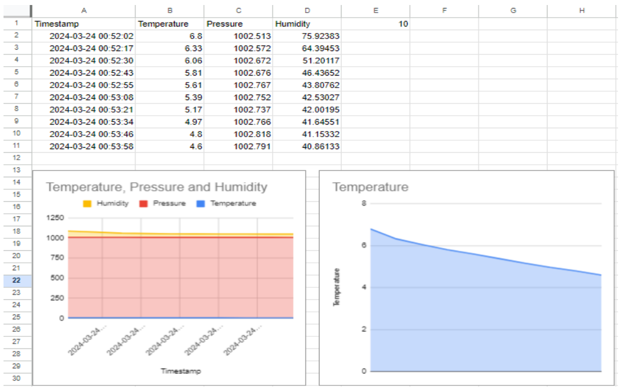


[Fig.11 IoT Architecture of the Prototype]

3. a snapshot of the output of the Python scripts, a snapshot of the web server with sensor data(website), a snapshot of the data as logged on the cloud and a snapshot of the obtained graph with regards to part c) above

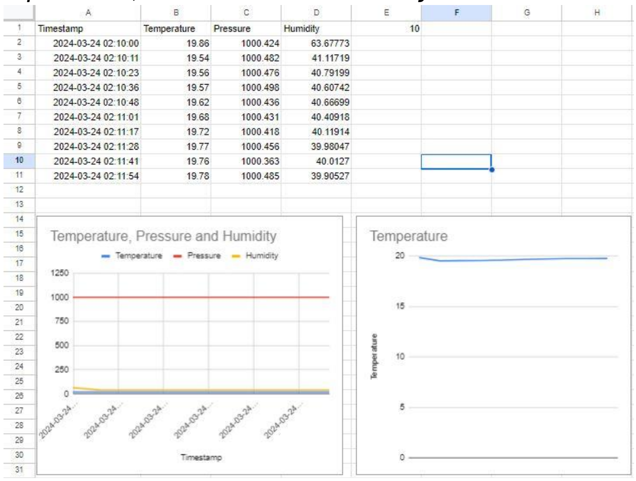
-A snapshot of the output of the Python Scrips(Fig 2, 5 and 8)
-A snapshot of the web server with sensor data(Fig 5)

-A snapshot of the data as logged on the cloud and a snapshot of the obtained graph with regards to part c



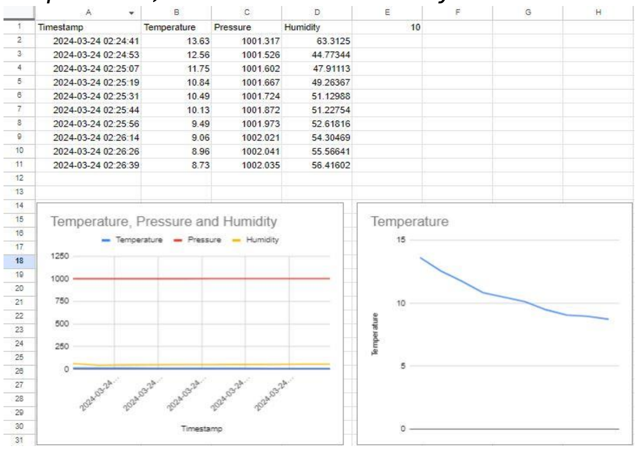
Timestamp	Temperature	Pressure	Humidity	10
2024-03-24 00:52:02	6.8	1002.513	75.92383	
2024-03-24 00:52:17	6.33	1002.572	64.39453	
2024-03-24 00:52:30	6.06	1002.672	51.20117	
2024-03-24 00:52:43	5.81	1002.676	46.43652	
2024-03-24 00:52:55	5.61	1002.767	43.80762	
2024-03-24 00:53:08	5.39	1002.752	42.53027	
2024-03-24 00:53:21	5.17	1002.737	42.00195	
2024-03-24 00:53:34	4.97	1002.766	41.64551	
2024-03-24 00:53:46	4.8	1002.818	41.15332	
2024-03-24 00:53:58	4.6	1002.791	40.86133	

[Fig.12 Temperature, Pressure and Humidity Results inside the Fridge]



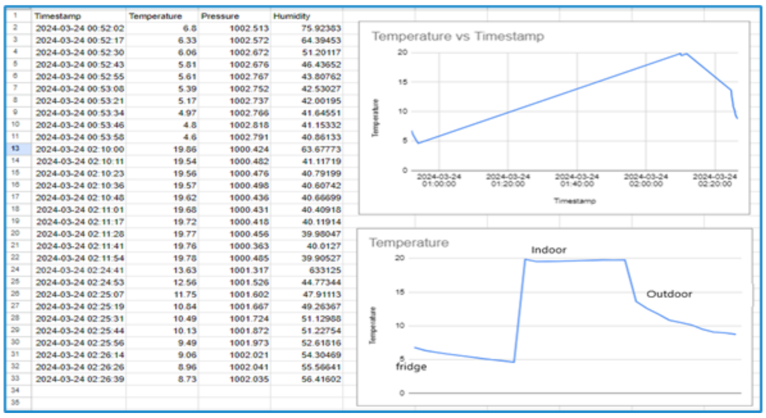
Timestamp	Temperature	Pressure	Humidity	10
2024-03-24 02:10:00	19.86	1000.424	63.67773	
2024-03-24 02:10:11	19.54	1000.482	41.11719	
2024-03-24 02:10:23	19.56	1000.476	40.79199	
2024-03-24 02:10:36	19.57	1000.498	40.60742	
2024-03-24 02:10:48	19.62	1000.436	40.66699	
2024-03-24 02:11:01	19.68	1000.431	40.40918	
2024-03-24 02:11:17	19.72	1000.418	40.11914	
2024-03-24 02:11:28	19.77	1000.456	39.98047	
2024-03-24 02:11:41	19.76	1000.363	40.0127	
2024-03-24 02:11:54	19.78	1000.485	39.90527	

[Fig.13 Temperature, Pressure and Humidity Results Indoor/Room]



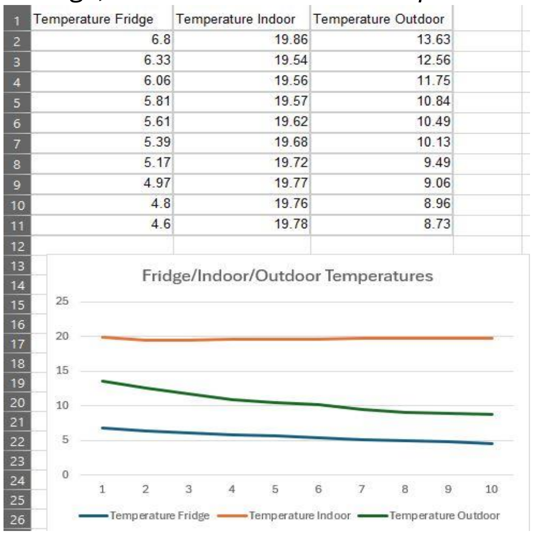
Timestamp	Temperature	Pressure	Humidity	10
2024-03-24 02:24:41	13.63	1001.317	63.3125	
2024-03-24 02:24:53	12.56	1001.526	44.77344	
2024-03-24 02:25:07	11.75	1001.602	47.91113	
2024-03-24 02:25:19	10.84	1001.667	49.26367	
2024-03-24 02:25:31	10.49	1001.724	51.12988	
2024-03-24 02:25:44	10.13	1001.872	51.22754	
2024-03-24 02:25:56	9.49	1001.973	52.61816	
2024-03-24 02:26:14	9.06	1002.021	54.30469	
2024-03-24 02:26:26	8.96	1002.041	55.56641	
2024-03-24 02:26:39	8.73	1002.035	56.41602	

[Fig.14 Temperature, Pressure and Humidity Results Outdoor]



	Timestamp	Temperature	Pressure	Humidity
2	2024-03-24 00:52:02	6.8	1002.513	75.92383
3	2024-03-24 00:52:17	6.33	1002.572	64.39453
4	2024-03-24 00:52:30	6.06	1002.672	51.20117
5	2024-03-24 00:52:43	5.81	1002.676	46.43652
6	2024-03-24 00:52:55	5.61	1002.767	43.80762
7	2024-03-24 00:53:08	5.39	1002.752	42.53027
8	2024-03-24 00:53:21	5.17	1002.737	42.00195
9	2024-03-24 00:53:34	4.97	1002.766	41.64551
10	2024-03-24 00:53:46	4.8	1002.818	41.15332
11	2024-03-24 00:53:58	4.6	1002.791	40.86133
13	2024-03-24 02:10:00	19.86	1000.424	63.67773
14	2024-03-24 02:10:11	19.54	1000.482	41.11719
15	2024-03-24 02:10:23	19.56	1000.476	40.79199
16	2024-03-24 02:10:36	19.57	1000.498	40.60742
17	2024-03-24 02:10:48	19.62	1000.436	40.66699
18	2024-03-24 02:11:01	19.68	1000.431	40.40918
19	2024-03-24 02:11:17	19.72	1000.418	40.11914
20	2024-03-24 02:11:28	19.77	1000.456	39.98047
21	2024-03-24 02:11:41	19.76	1000.363	40.0127
22	2024-03-24 02:11:54	19.78	1000.485	39.90527
24	2024-03-24 02:24:41	13.63	1001.317	633125
25	2024-03-24 02:24:53	12.56	1001.526	44.77344
26	2024-03-24 02:25:07	11.75	1001.602	47.91113
27	2024-03-24 02:25:19	10.84	1001.667	49.26367
28	2024-03-24 02:25:31	10.49	1001.724	51.12988
29	2024-03-24 02:25:44	10.13	1001.872	51.22754
30	2024-03-24 02:25:56	9.49	1001.973	52.61816
31	2024-03-24 02:26:14	9.06	1002.021	54.30469
32	2024-03-24 02:26:26	8.96	1002.041	55.56641
33	2024-03-24 02:26:39	8.73	1002.035	56.41602

[Fig.15 Overall of Fridge, Indoor and Outdoor Temperatures vs Timestamp]



	Temperature Fridge	Temperature Indoor	Temperature Outdoor
2	6.8	19.86	13.63
3	6.33	19.54	12.56
4	6.06	19.56	11.75
5	5.81	19.57	10.84
6	5.61	19.62	10.49
7	5.39	19.68	10.13
8	5.17	19.72	9.49
9	4.97	19.77	9.06
10	4.8	19.76	8.96
11	4.6	19.78	8.73

[Fig.16 Overall of Fridge, Indoor and Outdoor Temperatures Compared]

4. a discussion related to the testing part including a suitable decision-making approach

A structured and comprehensive approach is necessary when multiple tools are used to gather real-world data. The IoT prototype was tested in three different locations: the fridge, indoors, and outdoors. The choice of location is based on the effectiveness of the prototype in a cold environment(Fridge), a controlled environment(Room/Indoor) and a variable environment(Outdoor).

Conducting tests on a prototype across different atmospheric conditions and settings is essential to ensure its optimal performance and adaptability. We can confirm that the prototype functions correctly under various conditions by testing it thoroughly. Once the testing is completed, we can confidently declare our prototype as reliable and capable of withstanding any condition it may come across.

Additionally, another factor that can affect the readings is the connection, walls and structures might interfere with the wireless signal emitted by the hotspot, throughout the tests made we can check if the signal integrity can be disrupted luckily during the tests made there was no signal of disruption, however, while carrying out testing phase in the fridge we noticed that the hotspot device used to ensure connectivity needs to be as close as possible to the board.

Throughout the course of my testing methodology, I implemented three distinct temperature settings, specifically targeting the fridge, indoor environment, and outdoor conditions.

Concurrently, we established a connection between the cloud application and a Google sheet, which allowed us to continuously monitor the accuracy of the data transmitted, including the timestamp.

5. if this prototype was to be expanded to operate as an IoT air pollution monitoring system, briefly discuss three improvements/enhancements that would be necessary. Comment on how Artificial Intelligence can be useful and what tools would be appropriate

Air pollution leads to the existence of hazardous and harmful chemicals and gases present in the air. Changes in the nature of physical, chemical or biological air can lead to deterioration of the quality of life in the stratospheric environment. A current hot topic is carbon dioxide(CO2), the primary greenhouse gas fuelling global climate change. Various sensors need to be used to identify carbon dioxide gas present in the air, such as MQ-135, an air pollution gas sensor interfaced with Arduino uno and DHT11; it operates on low power, making it energy-efficient and suitable for battery-powered applications or continuous monitor systems.

The amount of air pollutants present in the atmosphere is influenced by various atmospheric conditions, such as wind speed, wind direction, relative humidity, and temperature. Air pressure also plays a significant role in the increase or decrease of pollution levels. During high-pressure systems, the air is more stagnant, which can cause pollution levels to rise. Conversely, during low-pressure systems, the weather tends to be damp and windy, leading to pollutants getting dispersed through rain or other weather conditions. This key factor gives us the importance of the placement, the device should be positioned in a balanced environment.

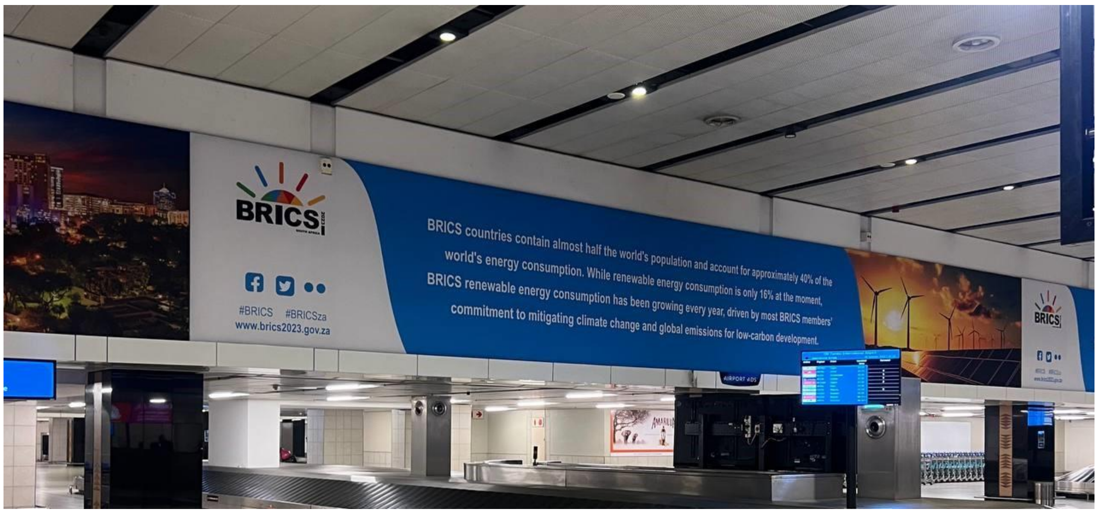
To ensure accurate and up-to-date monitoring and data collection, it's crucial to perform these tasks in real-time, when choosing an appropriate protocol, we need to consider the distance range; for example, LoRa is suitable for long-range applications, while for short-range applications, we can use Bluetooth. Another factor that goes hand in hand is power consumption and cost, which are relatively linked together. Less power consumption means fewer costs. Incorporating these features into a prototype not only enhances its efficiency but also positions it within the eco-friendly/sustainable category.

In an IoT environment, scalability and security are essential factors to consider. With the increasing amount of data being collected and stored, it is important to have an appropriate platform that can handle the data and protect it from security threats. To address this, we have selected Microsoft's Azure Synapse Analytics service.

Azure Synapse Analytics is an analytical platform that provides a workspace for data preparation, management, exploration, and warehousing. It offers an integrated experience that brings together big data and data warehousing. The platform provides a secure and scalable environment for data processing and storage, making it an ideal solution for IoT environments.

One of the key features of Azure Synapse Analytics is its incorporation of artificial intelligence to visualize data in a smarter way. This allows for more efficient and effective data analysis and helps users make better decisions based on insights gained from the data. With Azure Synapse Analytics, businesses can gain valuable insights from their IoT data and use this information to improve their operations and make better decisions.

The world is changing, and renewable energy and low carbon emissions are the future. These are reminders sent by BRICS countries in this slogan in an airport in China.


BRICS 2023 SOUTH AFRICA
#BRICS #BRICSza
www.brics2023.gov.za
BRICS countries contain almost half the world's population and account for approximately 40% of the world's energy consumption. While renewable energy consumption is only 16% at the moment, BRICS renewable energy consumption has been growing every year, driven by most BRICS members' commitment to mitigating climate change and global emissions for low-carbon development.
AIRPORT ADS
[Fig.17 BRICS Banner]

With this being said, European private banks have already started offering services to their customers to buy carbon credit, while other banks around the world have started to charge customers to pay for the carbon consumed, for example, flights. The prototype could be expanded to create a system for smartwatches(Smartwatch CO2 Tracker) or devices with MQ-135 sensors that are able to calculate the carbon footprint spent; the sensor, in order to be suitable, requires calibration to ensure accurate measurements and environmental factors such as temperature and humidity can influence the gas concentration in the readings. The creation of a solid algorithm is crucial in this context to translate the concertation of gases detected by the MQ-135 sensor into an estimate of carbon footprint, the device should be able to process data in real-time or near to provide feedback to the user and the optimization of the software is necessary to ensure that the device battery life is not significantly impacting while in use. Last but not least is the design of the user interface and data privacy; this can include daily or weekly summaries, tips for reducing emissions, progress tracking towards personal goals and implementation of robust data encryption because data privacy is crucial, especially when analysing behaviour patterns.

With the clear steps made by Artificial Intelligence in recent years, we can incorporate this technology to gain useful insights, such as identifying patterns and trends occurring in air pollution or detecting sudden spikes or peaks in pollutant levels. Therefore, we can enable AI to manage the air condition remotely.

Machine learning models such as LSTM(Long Short Term Memory/Time Series) can be deployed to forecast indoor and outdoor quality, this type of model is ideal for indoor infrastructure as the range of the area is not large compared to the outdoor, however, a detailed selection of features must be analysed before starting, positive results can detect high-risk particle such as carbon dioxide that can lead to a series of negative health effects such as nausea, headache and fatigue.

6. The printout of the source code you wrote with comments is in the Appendix.

Appendix

A) Source code Part A

```
#imports of necessary libraries and modules
#Hardware
from machine import Pin, I2C
import bme280
#Data Handling
import time

#Initialization of the I2C protocol
i2c=I2C(0, sda=Pin(0), scl=Pin(1), freq= 40000)

#instance of the bme280 with I2C initialized
bme = bme280.BME280(i2c=i2c)
#create a for loop 8 times to read the data
for i in range(8):

   #Reading data from sensors and storing the values on the variables temp, pressure and humidity.
   temp = bme.values[0]
   pressure = bme.values[1]
   humidity = bme.values[2]

   #concatenation of variables and strings to form a readable string
   reading = 'Current Temperature: |' + temp + '| Current Humidity: ' + humidity + '| Cureent Pressure: ' +
pressure

   #print the string with values
   print(reading)
   #pause the loop for 1 second
   time.sleep_ms(1000)
```

2) Source code Part B

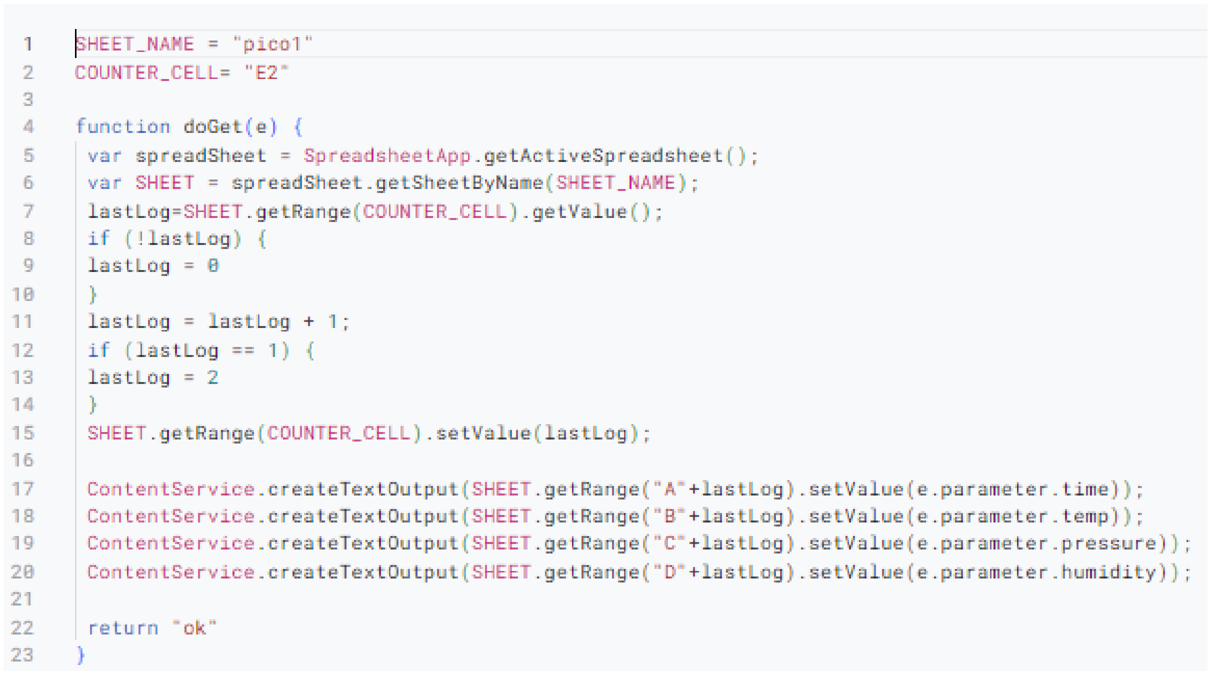
```
#imports of necessary libraries and modules
from machine import Pin, I2C
import bme280
from time import sleep
#Networking
import network
import socket

#Initialization of the I2C protocol
i2c=I2C(0, sda=Pin(0), scl=Pin(1), freq= 40000)
#instance of the bme280 with I2C initialized
bme = bme280.BME280(i2c=i2c)


#Hotspot credentials
ssid = 'iPhoneH'
password = 'qwerty123'

#function to connect to WLAN
def connect():
  wlan = network.WLAN(network.STA_IF)
  wlan.active(True)
  wlan.connect(ssid, password)

  while wlan.isconnected() == False:
    print('Waiting for connection...')
    sleep(1)
  ip = wlan.ifconfig()[0]
  print(f'Connected on {ip}')
  return ip

#function to open a socket
def open_socket(ip):

  address = (ip, 80)
  connection = socket.socket()
  connection.bind(address)
  connection.listen(1)
  print(connection)
  return connection

#create html template
def webpage(reading):

  html = f"""
      <!DOCTYPE html>
      <html>
      <head>
      <title>Pico W Temperature</title>
      <meta http-equiv='refresh' content='10'>
      <style>
        h1 {{
          text-align: center;
        }}
      </style>
      </head>
      <body>

      <h1>Readers</h1>
      <p>{reading}</p>

      </body>
      </html>
      """
  return str(html)

#function to send sensor data to a webpage
def serve(connection):
  while True:
    client = connection.accept()[0]
    request = client.recv(1024)
    request = str(request)

    #Reading data from sensors and storing the values on the variables temp, pressure and humidity
    temp = bme.values[0]
    pressure = bme.values[1]
    humidity = bme.values[2]

    reading = f'Current Temperature: {temp} | Current Humidity: {humidity} | Current Pressure: {pressure}'

    html = webpage(reading)
    client.send(html)

client.close()

#Implements try-except mechanism to connect to the network, open socket for data serving, and handle KeyboardInterrupt with device reset
try:
  ip = connect()
  connection = open_socket(ip)
  serve(connection)
except KeyboardInterrupt:
  machine.reset()
```

3) Source code Part C

```
#imports of necessary libraries and modules
#Hardware
import machine
from machine import Pin, I2C
import bme280
#Networking
import network
import urequests
#Data Handling
import time
import json
import gc

#Setting up wlan for the hotspot and initialize the LED as output device
wlan = network.WLAN(network.STA_IF)
board_led = machine.Pin("LED", machine.Pin.OUT)

#Initialization of the I2C protocol
i2c=I2C(0, sda=Pin(0), scl=Pin(1), freq= 40000)
#instance of the bme280 with I2C initialized
bme = bme280.BME280(i2c=i2c)

#Hotspot credentials
Wifi_ssid = 'iPhoneH'
Wifi_password = 'qwerty123'

#URLs for Google Script to log data and API for current time
SHE_URL                                                                 =
"https://script.google.com/macros/s/AKfycbywS64ieptzcryzjx6wB7QxyD9Jz439sbxb_UaGtCbAa
bqLm9VeHHUTBnkaXdGXiElD/exec"
TIME_URL = "https://timeapi.io/api/Time/current/zone?timeZone=Europe/London"

#function to get current time using http
def getCurrentTime():
   res = urequests.get(url=TIME_URL)
   time_data = json.loads(res.text)["dateTime"]
   res.close()
   return time_data

#function to connect to the hotspot
def connectWiFi():
   wlan.active(True)
   if not wlan.isconnected():
      wlan.connect(ssid, password)
      while not wlan.isconnected() and wlan.status() >= 0:
         print("Waiting to connect: ")
         time.sleep(1)
         board_led.value(not board_led.value())  # Toggle LED
      board_led.on()
      print(wlan.ifconfig())
   else:
      print("Wifi already connected...")
      #print current time
      print(getCurrentTime())

#function to send sensor data to the spreadsheet using the url in line 28
def sendToSpreadsheet(time, temp, pressure, humidity):
   try:
      url                                                                   =
f"{SCRIPT_URL}?time={time}&temp={temp}&pressure={pressure}&humidity={humidity}"
      print(url)
      res = urequests.get(url=url)
      res.close()
      gc.collect()
   except Exception as e:
      print("Error...", e)

#Connect to hotspot
connectWiFi()
```

```
#create a for loop 10 times to read the data from
the sensor and sent it to the spreadsheet
for i in range(10):

   value = bme.read_compensated_data()

   #Reading data from sensors, storing the values
on the variables and divide for meaningful units
   temp = value[0] / 100.0
   pressure = value[1] / 25600.0
   humidity = value[2] / 1024.0

   #get current time
   timestamp = getCurrentTime()

   #send the data to the spreadsheet
   sendToSpreadsheet(time=timestamp,
temp=temp,                    pressure=pressure,
humidity=humidity)
   #interval of 5 second between one reading and
the next one
   time.sleep(5)

#tunr off the LED light after the readings
board_led.off()
```

App Script



```
SHEET_NAME = "pico1"
COUNTER_CELL= "E2"

function doGet(e) {
 var spreadSheet = SpreadsheetApp.getActiveSpreadsheet();
 var SHEET = spreadSheet.getSheetByName(SHEET_NAME);
 lastLog=SHEET.getRange(COUNTER_CELL).getValue();
 if (!lastLog) {
 lastLog = 0
 }
 lastLog = lastLog + 1;
 if (lastLog == 1) {
 lastLog = 2
 }
 SHEET.getRange(COUNTER_CELL).setValue(lastLog);

 ContentService.createTextOutput(SHEET.getRange("A"+lastLog).setValue(e.parameter.time));
 ContentService.createTextOutput(SHEET.getRange("B"+lastLog).setValue(e.parameter.temp));
 ContentService.createTextOutput(SHEET.getRange("C"+lastLog).setValue(e.parameter.pressure));
 ContentService.createTextOutput(SHEET.getRange("D"+lastLog).setValue(e.parameter.humidity));

 return "ok"
}
```