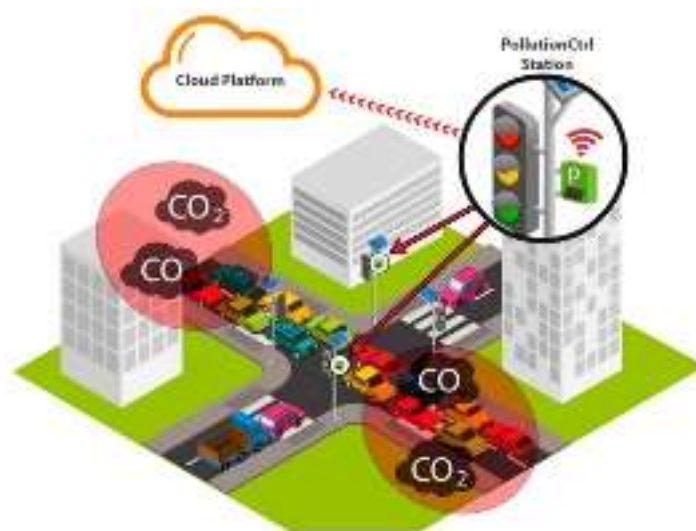


AIR QUALITY MONITORING

Phase 3: Development Part 1

- ❖ In this part you will begin building your project.
- ❖ Start building the IoT air quality monitoring system.
- ❖ Configure IoT devices (e.g., sensors for pollution levels, particulate matter) to measure air quality parameters.
- ❖ Develop a Python script on the IoT device to send collected data to the data-sharing platform.



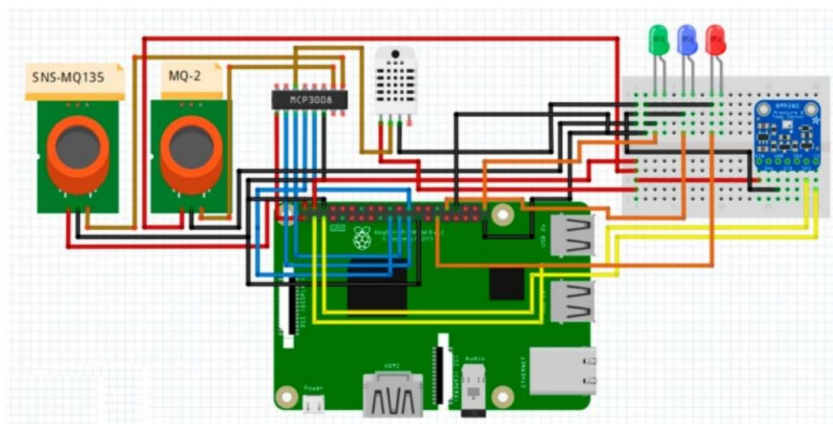
Introduction

The Internet of Things connects various devices, sensors, and systems to the internet, enabling them to interact and share data.

Python's user-friendly nature and vast library support make it an ideal choice for IoT projects. Let's explore how Python plays a crucial role in data collection, analysis, and device control in IoT applications.

Python in IoT Development

Python's popularity in the IoT realm is attributed to its ease of use, cross-platform compatibility, and extensive library ecosystem. Some of the key Python libraries and frameworks used in IoT development include:



❖ **MQTT (Message Queuing Telemetry Transport):**

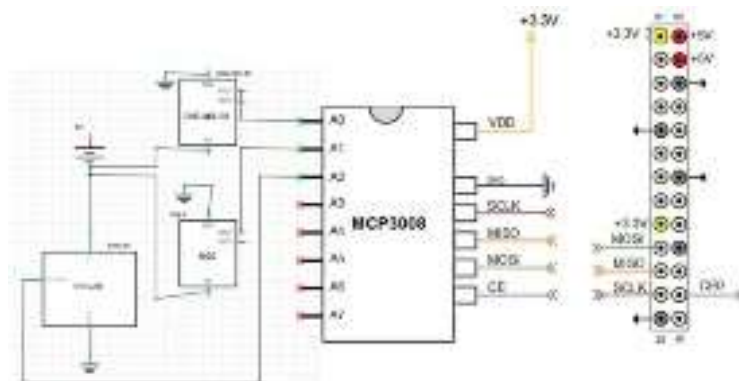
MQTT is a lightweight messaging protocol commonly used in IoT systems to facilitate communication between devices.

❖ **Requests:** This library simplifies making HTTP requests, which is essential for interfacing with cloud services and APIs.

❖ **NumPy and Pandas:** NumPy and Pandas are vital for data manipulation and analysis in IoT projects.

❖ **Matplotlib and Seaborn:** These libraries are used for data visualization, enabling developers to create insightful graphs and charts.

❖ **IoT Platforms and Frameworks:** Python is widely supported by popular IoT platforms and frameworks like Arduino, Raspberry Pi, and MicroPython, making it easier to develop IoT applications.



Data Collection with Python

Data collection is a fundamental aspect of IoT applications, where sensor data, environmental information, or user inputs are collected for analysis. Python's versatility makes it compatible with various data collection methods, such as reading sensor data, scraping web data, or receiving data from APIs.

```
# Example: Reading sensor data from a Raspberry Pi with Python
import Adafruit_DHT

# Set up the sensor type and pin
sensor = Adafruit_DHT.DHT11
pin = 4
# Read the sensor data
humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
# Display the results
if humidity is not None and temperature is not None:
    print(f'Temperature: {temperature}°C')
    print(f'Humidity: {humidity}%')
else:
    print('Failed to retrieve sensor data')
```

In this code snippet, we use the `Adafruit_DHT` library to read data from a DHT11 temperature and humidity sensor connected to a Raspberry Pi. Python's compatibility with various sensors simplifies the process of data collection in IoT projects.

Data Analysis and Visualization

Once data is collected, Python comes into play for data analysis and visualization. Libraries like NumPy and Pandas provide powerful tools for data manipulation and statistical analysis, while Matplotlib and Seaborn enable developers to create visually appealing graphs and charts.

```
# Example: Data analysis and visualization with Python
import pandas as pd
import matplotlib.pyplot as plt

# Sample data (replace this with your actual data)
data = {'Timestamp': ['2023-07-23', '2023-07-24', '2023-07-25'],
        'Temperature (°C)': [25.2, 26.5, 24.8],
        'Humidity (%)': [60, 55, 62]}
# Create a DataFrame
df = pd.DataFrame(data)
# Perform data analysis (e.g., calculating mean, max, etc.)
average_temperature = df['Temperature (°C)'].mean()
maximum_humidity = df['Humidity (%)'].max()
# Create a line plot to visualize temperature trends
plt.plot(df['Timestamp'], df['Temperature (°C)'])
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.title('Temperature Trends')
plt.xticks(rotation=45)
plt.show()
# Display the data analysis results
print(f'Average Temperature: {average_temperature}°C')
print(f'Maximum Humidity: {maximum_humidity}%')
```

In this code snippet, we use Pandas for data manipulation and Matplotlib to create a line plot showing temperature trends over time. Python's data analysis and visualization capabilities aid in gaining valuable insights from IoT data.

Device Control with Python

Python's versatility extends to controlling IoT devices, enabling developers to send commands and control physical devices from software applications. Whether it's turning on a light, controlling a motor, or adjusting the temperature, Python facilitates device control in IoT projects

```
# Example: Controlling a LED with Python and Raspberry Pi
import RPi.GPIO as GPIO
import time

# Set up the GPIO pin
led_pin = 18
GPIO.setmode(GPIO.BCM)
GPIO.setup(led_pin, GPIO.OUT)
# Turn on the LED
GPIO.output(led_pin, GPIO.HIGH)
time.sleep(3)
# Turn off the LED
GPIO.output(led_pin, GPIO.LOW)
# Clean up the GPIO
GPIO.cleanup()
```

In this code snippet, we use the Raspberry Pi's GPIO library to control an LED connected to a GPIO pin. Python's support for hardware interfacing simplifies the process of device control in IoT projects. Python's adaptability, rich library support, and user-friendly syntax make it an ideal language for Internet of Things (IoT) projects. In this article, we explored how Python is employed in IoT applications, specifically focusing on data collection, analysis, and device control. Python's role in IoT development extends beyond the examples provided here, as its versatility allows developers to explore various use cases and unleash the full potential of IoT technologies.

Whether you're a seasoned IoT developer or just starting with IoT projects, Python offers a reliable and robust environment to build innovative solutions that connect the physical and digital worlds. Embrace the power of Python and embark on your IoT journey to shape a smarter, interconnected future.

Conclusion

Python's adaptability, rich library support, and user-friendly syntax make it an ideal language for Internet of Things (IoT) projects. In this article, we explored how Python is employed in IoT applications, specifically focusing on data collection, analysis, and device control. Python's role in IoT development extends beyond the examples provided here, as its versatility allows developers to explore various use cases and unleash the full potential of IoT technologies.

Whether you're a seasoned IoT developer or just starting with IoT projects, Python offers a reliable and robust environment to build innovative solutions that connect the physical and digital worlds. Embrace the power of Python and embark on your IoT journey to shape a smarter, interconnected future.