Phase 3: NOISE POLLUTION

Step 1: Assemble the Hardware :

1. **Select IoT Noise Sensors**: Choose suitable noise sensors capable of measuring noise levels in decibels (dB). Ensure the selected sensors are compatible with your IoT platform and can transmit data reliably.
2. **IoT Platform**: Decide on the IoT platform you'll be using for data transmission. This could be a cloud-based platform like AWS IoT, Google Cloud IoT, or an on-premises solution.
3. **Connectivity**: Set up the required connectivity options, such as Wi-Fi, cellular, or LoRa, to connect the sensors to the IoT platform.

Step 2: Deploy IoT Noise Sensors

1. **Identify Deployment Locations**: Determine the public areas where you want to monitor noise pollution. These locations should be representative of the areas affected by noise pollution.
2. **Install Sensors**: Securely install the IoT noise sensors at the selected locations. Ensure they are protected from weather conditions and vandalism, as well as powered appropriately (e.g., using batteries or a power source).
3. **Network Configuration**: Configure the sensors to connect to the selected IoT platform. This may involve configuring Wi-Fi or cellular settings, setting up security protocols, and establishing communication with the platform.

Step 3: Develop Python Script

1. **Programming Environment**: Set up a development environment for writing and testing Python scripts. You can use popular IDEs like Visual Studio Code, PyCharm, or Jupyter Notebook.
2. **Sensor Data Collection**: Write Python code on the IoT sensors to collect noise level data. Most noise sensors provide APIs or libraries for data collection. Make sure you follow the sensor manufacturer's documentation for data acquisition.
3. **Data Preprocessing**: Process and format the collected data as needed. This may include filtering, averaging, or adjusting the data for better accuracy.
4. **Data Transmission**: Develop Python scripts to transmit the processed noise level data to the IoT platform. Utilize libraries or SDKs provided by the platform for secure and reliable data transmission. For example, if you're using MQTT for data transport, you can use libraries like Paho MQTT.
5. **Security**: Implement security measures to protect the data in transit. Use encryption, secure protocols, and authentication methods to ensure data integrity and privacy.
6. **Error Handling**: Include error-handling mechanisms in your Python script to manage network connectivity issues, sensor failures, and other potential problems that may arise during data transmission.

7. **Real-Time Capabilities**: Ensure that the script can transmit data in real-time or near real-time to provide up-to-date information on noise levels.

Step 4: Testing and Validation

1. Test the Python script on the IoT noise sensors to ensure that it collects and transmits data accurately.
2. Validate the data on the IoT platform to confirm that it is received and stored correctly.

Step 5: Monitoring and Maintenance

1. Implement remote monitoring and alerting to detect sensor malfunctions or connectivity issues.
2. Establish a maintenance schedule for sensor calibration and any necessary repairs.
3. Continuously monitor the IoT-enabled Noise Pollution Monitoring system for data accuracy and system health.

By following these steps, you can deploy IoT noise sensors in public areas, develop a Python script for real-time data transmission, and begin collecting noise level data for your Noise Pollution Monitoring system. This data will serve as the foundation for further analysis and decision-making regarding noise pollution management.

PROGRAM:

```
import time
import random
import paho.mqtt.client as mqtt

# MQTT Broker settings
broker_address = "mqtt.example.com"  # Replace with your MQTT broker's address
broker_port = 1883
topic = "noise_levels"  # Topic where noise data will be published

# Simulate noise data (replace this with actual sensor data)
def get_noise_level():
    return round(random.uniform(50, 90), 2)  # Simulated noise levels between 50 and 90 dB

# MQTT callback functions
def on_connect(client, userdata, flags, rc):
    if rc == 0:
```

```
        print("Connected to MQTT broker")
    else:
        print("Failed to connect, return code: " + str(rc))

def on_publish(client, userdata, mid):
    print("Data published to MQTT broker")

# Initialize the MQTT client
client = mqtt.Client()
client.on_connect = on_connect
client.on_publish = on_publish

# Connect to the MQTT broker
client.connect(broker_address, broker_port, keepalive=60)

# Start the MQTT loop (non-blocking)
client.loop_start()

try:
    while True:
        # Get noise level data
        noise_level = get_noise_level()

        # Create a payload with the noise level
        payload = f"{{\"noise_level\": {noise_level}}}"

        # Publish the data to the MQTT topic
        client.publish(topic, payload)

        print(f"Noise level data sent: {payload}")

        # Wait for a specified interval before sending the next data
```

```
        time.sleep(10)  # Adjust as needed
except KeyboardInterrupt:
    print("Script terminated by user")

# Disconnect from the MQTT broker
client.disconnect()
```