Experiment No 2

AIM: To simulate sensor data generation and collection.

Objective: To simulate sensor data generation and collection for IoT applications. We will:

1. Generate random sensor data for parameters like temperature and humidity.
2. Collect the generated data in real-time.
3. Save the data to a file for further analysis.

Tools Used: Google Colab, Python libraries (random)

Theory:
The Internet of Things (IoT) is a transformative concept, involving a network of interconnected devices that communicate and exchange data to enable smart environments. Sensor data generation is a fundamental aspect of IoT systems, where physical parameters such as temperature, humidity, and pressure are measured by sensors and transmitted to processing units or the cloud. This data is essential for analysis, decision-making, and actuation in applications ranging from home automation to industrial monitoring.

Simulating sensor data enables learners to replicate the behavior of IoT devices without the need for physical sensors. Using Python, we can generate pseudo-random data mimicking real-world sensor readings, which helps students understand IoT data streams and their processing. This approach is highly advantageous for prototyping and learning as it eliminates the dependency on hardware.

The use of the *random* module in Python allows us to generate realistic, variable readings within a defined range. For example, temperature readings can be emulated as floating-point values within a typical range for environmental conditions. Simulated data is stored and processed for tasks such as visualization, statistical analysis, or transmission over IoT communication protocols like MQTT or HTTP.

This experiment demonstrates the basics of creating a simulated IoT environment, emphasizing the importance of data generation, collection, and understanding real-world constraints like data variability and trends.

Program Code:

```
import random
import time
import json
# Function to simulate sensor data generation
def generate_sensor_data():
  """
  Simulates data for temperature and humidity sensors.
  Returns a dictionary containing the sensor readings.
  """
  temperature = round(random.uniform(20.0, 30.0), 2)  # Random temperature between 20°C and 30°C
  humidity = round(random.uniform(40.0, 60.0), 2)     # Random humidity between 40% and 60%
  timestamp = time.strftime("%Y-%m-%d %H:%M:%S")      # Current timestamp
  return {
      "timestamp": timestamp,
      "temperature": temperature,
      "humidity": humidity
  }
# Function to collect sensor data and save to a file
def collect_sensor_data(duration, interval, output_file):
  """
  Collects sensor data for a given duration and interval, and saves it to a file.
  Args:
  - duration: Total duration in seconds for data collection.
  - interval: Time interval (in seconds) between data points.
  - output_file: File to save the collected data in JSON format.
  """
  collected_data = []
  start_time = time.time()
  print("Starting sensor data collection...\n")
  while time.time()-start_time < duration:
    # Generate sensor data
    sensor_data = generate_sensor_data()

    # Display the generated data
    print(f"Collected Data: {sensor_data}")
        # Append to the collection
    collected_data.append(sensor_data)
        # Wait for the specified interval
    time.sleep(interval)
  # Save the data to a file
  with open(output_file, 'w') as file:
    json.dump(collected_data, file, indent=4)
    print(f"\nSensor data collection completed. Data saved to {output_file}")
# Main program
if __name__ == "__main__":
  # Define simulation parameters
  duration = 30  # Collect data for 30 seconds
  interval = 5   # Collect data every 5 seconds
```

```
output_file = "sensor_data.json"

# Start the data collection

collect_sensor_data(duration, interval, output_file)
```

Explanation and Logic of Code

1. **Sensor Data Simulation**:
    - Randomly generates values for **temperature** (20.0–30.0 °C) and **humidity** (40.0–60.0%) to mimic real-world sensor readings.
    - Includes a timestamp for real-time tracking.
2. **Data Collection**:
    - Runs a loop for a specified duration.
    - Collects data at defined intervals using time.sleep(interval) to simulate real-time behavior.
    - Accumulates the data in a list for structured storage.
3. **Data Storage**:
    - At the end of data collection, the sensor readings are saved in a **JSON file** for analysis and visualization.
4. **Dynamic Behavior**:
    - The program can adapt to different durations and intervals, making it versatile for various simulation needs.

Message Flow

Below is the message flow describing the interaction between system components:

Flowchart:

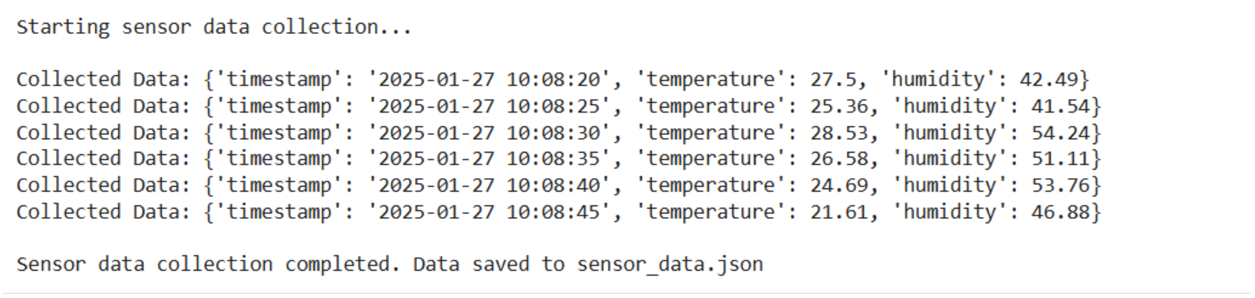
Start Program
→ Initialize Parameters (duration, interval, output file).
→ Begin Data Collection Loop:

- Generate sensor data.
- Append data to collection.
- Print and store data. → End Loop After Duration.
  → Save Data to JSON File.
  → Stop Program.

Observation Table:

Timestamp	Temperature (°C)	Humidity (%)	Remarks
2025-01-27 10:08:20	27.5	42.49	Normal readings.
2025-01-27 10:08:25	25.36	41.54	Slight drop in temperature.
2025-01-27 10:08:30	28.53	54.24	Humidity increased.
2025-01-27 10:08:35	26.58	51.11	Stable conditions.
2025-01-27 10:08:40	24.69	53.76	Drop in Temp.
2025-01-27 10:08:45	21.61	46.88	Normal readings.

Outcome :



```
Starting sensor data collection...

Collected Data: {'timestamp': '2025-01-27 10:08:20', 'temperature': 27.5, 'humidity': 42.49}
Collected Data: {'timestamp': '2025-01-27 10:08:25', 'temperature': 25.36, 'humidity': 41.54}
Collected Data: {'timestamp': '2025-01-27 10:08:30', 'temperature': 28.53, 'humidity': 54.24}
Collected Data: {'timestamp': '2025-01-27 10:08:35', 'temperature': 26.58, 'humidity': 51.11}
Collected Data: {'timestamp': '2025-01-27 10:08:40', 'temperature': 24.69, 'humidity': 53.76}
Collected Data: {'timestamp': '2025-01-27 10:08:45', 'temperature': 21.61, 'humidity': 46.88}

Sensor data collection completed. Data saved to sensor_data.json
```

Conclusion:

Simulating sensor data generation and collection is a foundational exercise in understanding IoT systems. This practical approach helps to grasp the concepts of real-time data acquisition, storage, and processing. By extending the simulation with tasks like visualization, anomaly detection, or cloud integration, We can bridge the gap between theory and real-world applications, preparing them for advanced IoT challenges

Homework Assigned:

AIM: To simulate sensor data generation and collection.

Objective: To simulate sensor data generation and collection for IoT applications. We will:

1. Generate random sensor data for parameters like temperature and humidity.
2. Collect the generated data in real-time.
3. Save the data to a file for further analysis.

Program Code:

```
import random

import time

import json

# Function to simulate sensor data generation

def generate_sensor_data():

    """

    Simulates data for temperature, humidity, pressure, and light intensity sensors.

    Returns a dictionary containing the sensor readings.

    """

    temperature = round(random.uniform(20.0, 30.0), 2)  # Random temperature between 20°C and 30°C

    humidity = round(random.uniform(40.0, 60.0), 2)     # Random humidity between 40% and 60%

    pressure = round(random.uniform(900.0, 1100.0), 2)  # Random pressure between 900 and 1100 hPa

    light_intensity = round(random.uniform(300.0, 800.0), 2)  # Random light intensity between 300-800

    timestamp = time.strftime("%Y-%m-%d %H:%M:%S")      # Current timestamp

    return {

        "timestamp": timestamp,

        "temperature": temperature,

        "humidity": humidity,
        "pressure": pressure,
        "light_intensity": light_intensity
    }
# Function to collect sensor data and save to a file
def collect_sensor_data(duration, interval, output_file):
    """
    Collects sensor data for a given duration and interval, and saves it to a file.
    Args:
    - duration: Total duration in seconds for data collection.
    - interval: Time interval (in seconds) between data points.
    - output_file: File to save the collected data in JSON format.
    """
    collected_data = []
    start_time = time.time()
    print("Starting sensor data collection...\n")
    while time.time() - start_time < duration:
        # Generate sensor data
        sensor_data = generate_sensor_data()
        # Display the generated data
        print(f"Collected Data: {sensor_data}")
        # Append to the collection
        collected_data.append(sensor_data)
        # Wait for the specified interval
        time.sleep(interval)
    # Save the data to a file
    with open(output_file, 'w') as file:
        json.dump(collected_data, file, indent=4)
    print(f"\nSensor data collection completed. Data saved to {output_file}")
# Main program
if __name__ == "__main__":
    # Define simulation parameters
    duration = 30  # Collect data for 30 seconds
    interval = 5   # Collect data every 5 seconds
    output_file = "extended_sensor_data.json"
    # Start the data collection
    collect_sensor_data(duration, interval, output_file)
```

Explanation and Logic of Code:

1. **Sensor Data Simulation**:
    - **Temperature**: Simulated between 20.0–30.0°C.
    - **Humidity**: Simulated between 40.0–60.0%.
    - **Pressure**: Added a new sensor to simulate atmospheric pressure between 900 and 1100 hPa.
    - **Light Intensity**: Added a new sensor to simulate light intensity between 300 and 800 lux.
    - Each reading includes a **timestamp** for tracking.
2. **Data Collection**:
    - The program collects data for a **specified duration** and at regular **intervals**.
    - Data is accumulated in a list and displayed in real-time.

3. **Data Storage**:
    - At the end of the collection, data is stored in a **JSON file**, making it easy to analyze and visualize.
4. **Dynamic Parameters**:
    - Users can modify the duration, interval, and output file name.

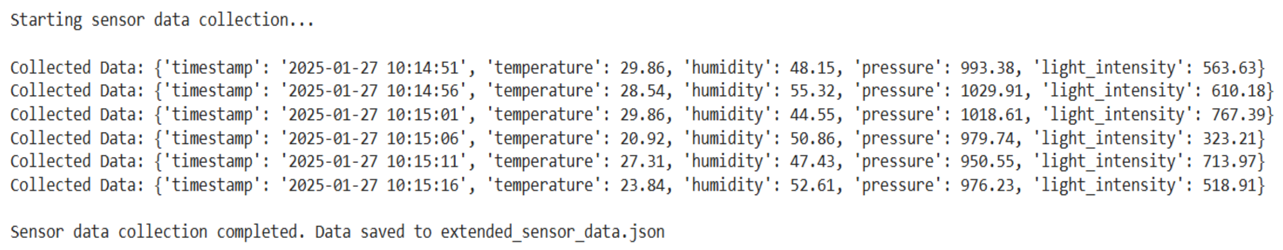
Message Flow:

1. **Start Program**:
    - Parameters are initialized: duration, interval, output file.
2. **Data Collection Loop**:
    - Generates sensor data for all sensors.
    - Appends data to the collection and displays in real-time.
    - Waits for the interval before the next reading.
3. **Save to File**:
    - After the loop ends, data is saved to a JSON file.
4. **End Program**:
    - Displays a completion message.

Observation Table:

Timestamp	Temperature (°C)	Humidity (%)	Pressure (hPa)	Light Intensity	Remark
2025-01-27 10:14:51	29.86	48.15	993.38	563.63	Normal conditions
2025-01-27 10:14:56	28.54	55.32	1029.91	610.18	Increased pressure, stable light
2025-01-27 10:15:01	29.86	44.55	1018.61	767.39	High light intensity
2025-01-27 10:15:06	20.92	50.86	979.74	323.21	Drop in temperature and light
2025-01-27 10:15:11	27.31	47.43	950.55	713.97	Moderate conditions
2025-01-27 10:15:16	23.84	52.61	976.23	518.91	Stable but cooler temperature

Outcome :



```
Starting sensor data collection...

Collected Data: {'timestamp': '2025-01-27 10:14:51', 'temperature': 29.86, 'humidity': 48.15, 'pressure': 993.38, 'light_intensity': 563.63}
Collected Data: {'timestamp': '2025-01-27 10:14:56', 'temperature': 28.54, 'humidity': 55.32, 'pressure': 1029.91, 'light_intensity': 610.18}
Collected Data: {'timestamp': '2025-01-27 10:15:01', 'temperature': 29.86, 'humidity': 44.55, 'pressure': 1018.61, 'light_intensity': 767.39}
Collected Data: {'timestamp': '2025-01-27 10:15:06', 'temperature': 20.92, 'humidity': 50.86, 'pressure': 979.74, 'light_intensity': 323.21}
Collected Data: {'timestamp': '2025-01-27 10:15:11', 'temperature': 27.31, 'humidity': 47.43, 'pressure': 950.55, 'light_intensity': 713.97}
Collected Data: {'timestamp': '2025-01-27 10:15:16', 'temperature': 23.84, 'humidity': 52.61, 'pressure': 976.23, 'light_intensity': 518.91}

Sensor data collection completed. Data saved to extended_sensor_data.json
```

Conclusion:

Simulating sensor data generation and collection with additional sensors (pressure and light intensity) enhances understanding of IoT systems. This exercise demonstrates how real-time data acquisition and storage can be extended for multi-sensor environments. The approach provides a strong foundation for further tasks such as anomaly detection, visualization, and cloud integration, bridging theoretical concepts with practical applications.