Project:

Smart water Managment

TEAM MEMBER:

T. Gowtham
S. Siva
A. Periyasamy
M. RajeshKumar

Smart water management is an essential aspect of modern urban planning and sustainability. To develop a smart water management system, you can consider the following key components:

Sensor Technology:

✓ Implement a network of sensors to monitor various aspects of the water system, including water quality, flow rates, pressure, and leak detection. These sensors can provide real-time data to a central control system.

Data Analytics:

✓ Use advanced data analytics and machine learning algorithms to process the data collected by sensors. This can help in identifying patterns, anomalies, and optimizing water usage.

IoT and Connectivity:

✓ Ensure that the sensors are connected through the Internet of Things (IoT) to gather data efficiently. This data should be transmitted securely to a central server for analysis.

Cloud Computing:

✓ Use cloud-based platforms for data storage and processing. Cloud computing can handle large volumes of data and provide scalability and accessibility.

Central Control System:

✓ Develop a central control system or software that can analyze the data and make real-time decisions. It should be capable of adjusting water distribution, identifying leaks, and optimizing water treatment processes.

Water Quality Monitoring:

✓ Implement a water quality monitoring system to ensure the safety and quality of water. This involves monitoring for contaminants and ensuring compliance with water quality standards.

Leak Detection:

✓ Integrate leak detection algorithms to identify and locate leaks in the water distribution network promptly. This can help reduce water wastage.

Demand Forecasting:

✓ Use historical data and predictive analytics to forecast water demand accurately. This can assist in proactive management of water resources.

Smart Metering:

✓ Install smart water meters to monitor individual water consumption in homes and businesses. This promotes water conservation and helps users understand their water usage patterns.

User Engagement:

✓ Educate and engage the community in water conservation efforts through mobile apps, websites, and alerts about water-saving practices.

Remote Control:

✓ Enable remote control of water infrastructure to make real-time adjustments, such as valve operations and pressure regulation.

Resilience and Redundancy:

✓ Build redundancy and resilience into the system to ensure continuous water supply, even in case of system failures or emergencies.

Regulatory Compliance:

✓ Ensure that the system complies with relevant water regulations and standards.

Data Security:

✓ Implement robust security measures to protect the data and the control system from cyber threats.

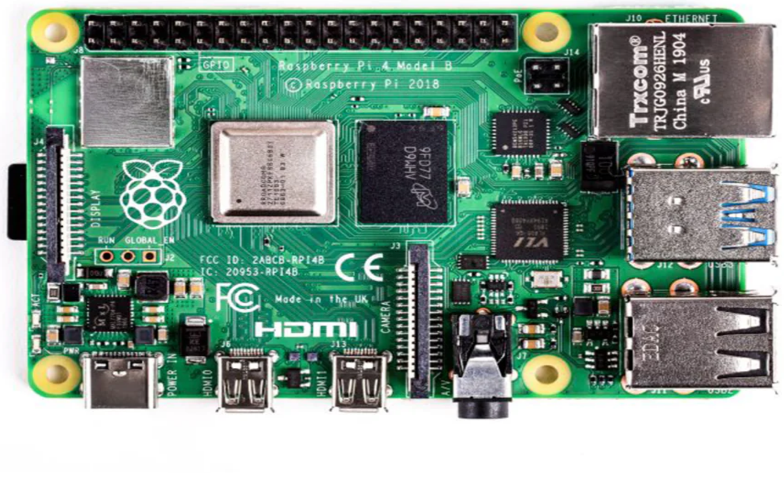
Integration with Other Systems:

✓ Integrate the smart water management system with other urban management systems, like traffic control and energy management, for a holistic approach to urban sustainability.

Maintenance and Training:

✓ Develop a maintenance plan and provide training for personnel to ensure the continued operation and optimization of the system.

Developing a smart water management system using a Raspberry Pi can be a cost-effective and versatile approach. Here are the steps and components to consider:



Raspberry Pi Setup:

✓ Start with a Raspberry Pi board (e.g., Raspberry Pi 4).
Set up the Raspberry Pi with an operating system (e.g., Raspberry Pi OS).
✓ Connect necessary peripherals (keyboard, mouse, monitor) or access it remotely via SSH.

Sensors and Hardware:

✓ Choose appropriate sensors for water quality, flow, pressure, and any other parameters you want to monitor.
✓ Connect sensors to the Raspberry Pi using GPIO pins or compatible interfaces (e.g., I2C or SPI).
Consider adding a relay module to control pumps or valves remotely.

Data Collection:

✓ Write Python scripts to read data from the sensors and collect it at regular intervals.
✓ Store the data locally on the Raspberry Pi or transmit it to a central server or cloud for further analysis.

Data Processing:

✓ Develop Python scripts for data processing and analysis, including anomaly detection, trend analysis, and decision-making algorithms.
✓ You can use libraries like NumPy, Pandas, or custom code for data manipulation.

User Interface:
✓ Create a web-based or mobile user interface for users to monitor water data and control devices.
✓ Frameworks like Flask, Django, or HTML/CSS/JavaScript can be used for the UI.

IoT Connectivity:
✓ Ensure your Raspberry Pi has internet connectivity via Wi-Fi or Ethernet.
✓ Use MQTT or other IoT protocols to communicate with remote devices or a central server.

Database:
✓ Set up a database to store historical data. SQLite or databases like MySQL or PostgreSQL can be used for local storage.

Security:
✓ Implement security measures to protect your Raspberry Pi and data.
✓ This includes using secure communication protocols and access control.

Automation:
✓ Implement automation routines for adjusting valves or pumps based on sensor data.
✓ Ensure that you have fail-safes and redundancy in case of system failures.

Remote Access:
✓ Set up secure remote access to your Raspberry Pi for maintenance and updates.

Power Management:

✓ Consider using uninterruptible power supplies (UPS) to ensure the system remains operational during power outages.

Testing and Calibration:

✓ Thoroughly test the system, calibrate sensors, and fine-tune algorithms for accurate data collection and control.

Scalability:

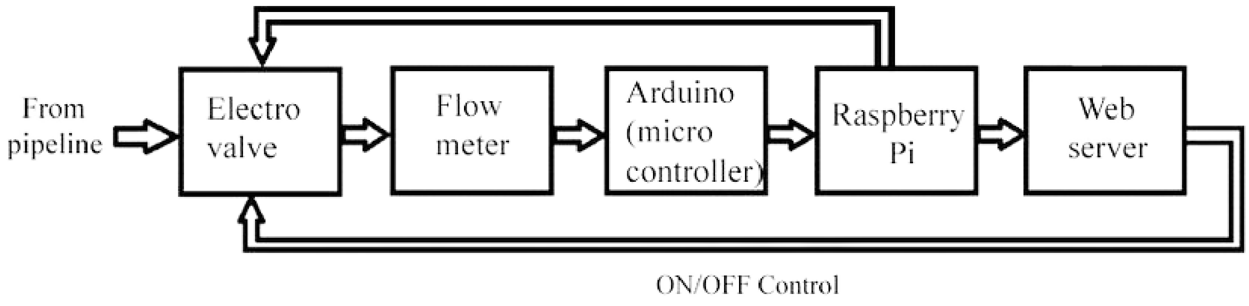
✓ Plan for scalability as your smart water management system expands. You may need to add more Raspberry Pi units or other devices.

Documentation and Support:

✓ Document your system setup and provide user manuals for maintenance and troubleshooting.

Data Visualization and Reporting:

✓ Implement data visualization tools or dashboards to help users understand water usage patterns and trends.

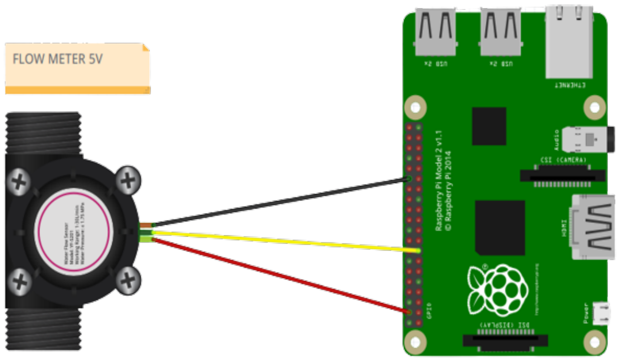


Building an IoT water consumption monitoring system involves several components, including hardware, software, and network infrastructure. Below, I'll outline the basic steps to get you started on creating such a system:

Hardware Components:

1.Water Flow Sensors:

✓ Select the appropriate flow sensors to measure water consumption accurately. These sensors can be attached to the water supply line to monitor flow rates.



2. Microcontroller or Single-Board Computer:

✓ Choose a microcontroller (e.g., Arduino, Raspberry Pi) to interface with the flow sensors and handle data processing. Ensure it has the necessary connectivity options like Wi-Fi, Ethernet, or cellular.

3. Power Supply:

✓ Depending on the deployment location, you may need a reliable power source. In some cases, battery-powered solutions are more practical.

4. Enclosure:

✓ Protect your hardware components from environmental factors by using suitable enclosures.

5. Internet Connectivity:

✓ Establish a connection method (Wi-Fi, Ethernet, or cellular) to transmit data from the monitoring system to the cloud or a central server.

Software Components:

1. Firmware:

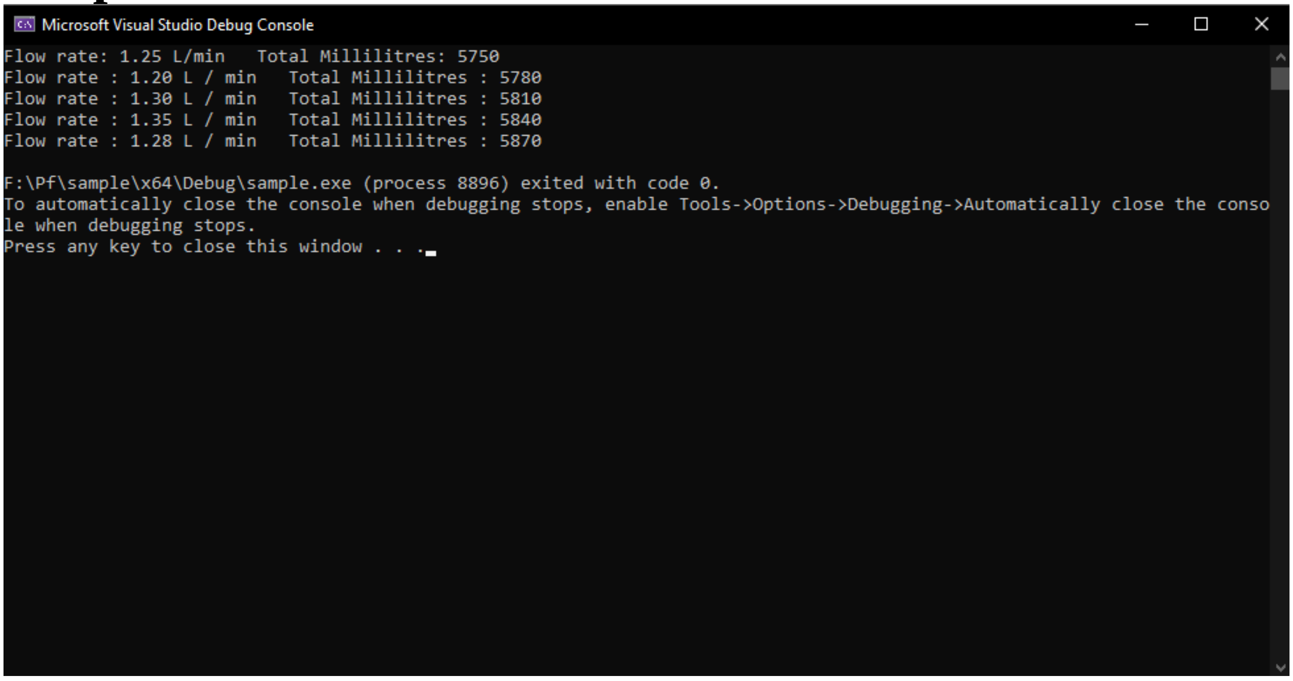
✓ Develop or configure firmware for the microcontroller to read data from the flow sensors, process it, and send it to the cloud. You can use programming languages like C/C++ (for Arduino) or Python (for Raspberry Pi).

Frimware To Measure the Water Meter generated data using c++:

```
const int flowSensorPin = 2; // Connect the flow sensor to a digital pin
```

```
volatile int pulseCount = 0;
float flowRate = 0.0; // Flow rate in liters per minute
unsigned int flowMilliLitres = 0;
unsigned long totalMilliLitres = 0;

unsigned long previousMillis = 0;
const long interval = 1000; // Update interval in milliseconds

void setup() {
  pinMode(flowSensorPin, INPUT);
  attachInterrupt(digitalPinToInterrupt(flowSensorPin), pulseCounter,
FALLING);
  Serial.begin(9600);
}

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    detachInterrupt(digitalPinToInterrupt(flowSensorPin));

    flowRate = (1000.0 / (float(interval) / 1000.0)) / pulseCount;
    flowMilliLitres = (flowRate / 60) * 1000;

    totalMilliLitres += flowMilliLitres;

    Serial.print("Flow rate: ");
    Serial.print(flowRate);
    Serial.print(" L/min\tTotal Millilitres: ");
    Serial.println(totalMilliLitres);

    pulseCount = 0;
    previousMillis = currentMillis;
    attachInterrupt(digitalPinToInterrupt(flowSensorPin), pulseCounter,
FALLING);
  }
}

void pulseCounter() {
  pulseCount++;
}
```

Output:



Microsoft Visual Studio Debug Console

```
Flow rate: 1.25 L/min   Total Millilitres: 5750
Flow rate : 1.20 L / min   Total Millilitres : 5780
Flow rate : 1.30 L / min   Total Millilitres : 5810
Flow rate : 1.35 L / min   Total Millilitres : 5840
Flow rate : 1.28 L / min   Total Millilitres : 5870

F:\Pf\sample\x64\Debug\sample.exe (process 8896) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

2. Cloud Platform:

✓ Choose a cloud platform to store and manage the data. Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure are popular options. Set up data storage and APIs to receive data from your devices.

✓To connect a Raspberry Pi to the AWS Cloud and send data from a sensor, you can follow these general steps:

Set up AWS Account:

✓If you don't have one, create an AWS account.
✓Navigate to the AWS IoT Core service and create an AWS IoT Thing.

Set up Raspberry Pi:

✓Ensure your Raspberry Pi is set up with Raspbian or a compatible operating system.
✓Install the AWS IoT SDK for Python on your Raspberry Pi using pip: pip install AWSIoTPythonSDK.

Create IoT Thing on AWS:

✓Create an IoT Thing in the AWS IoT Core.
✓Generate and download security certificates (X.509) for the Thing. You'll need the private key, public key, and certificate.

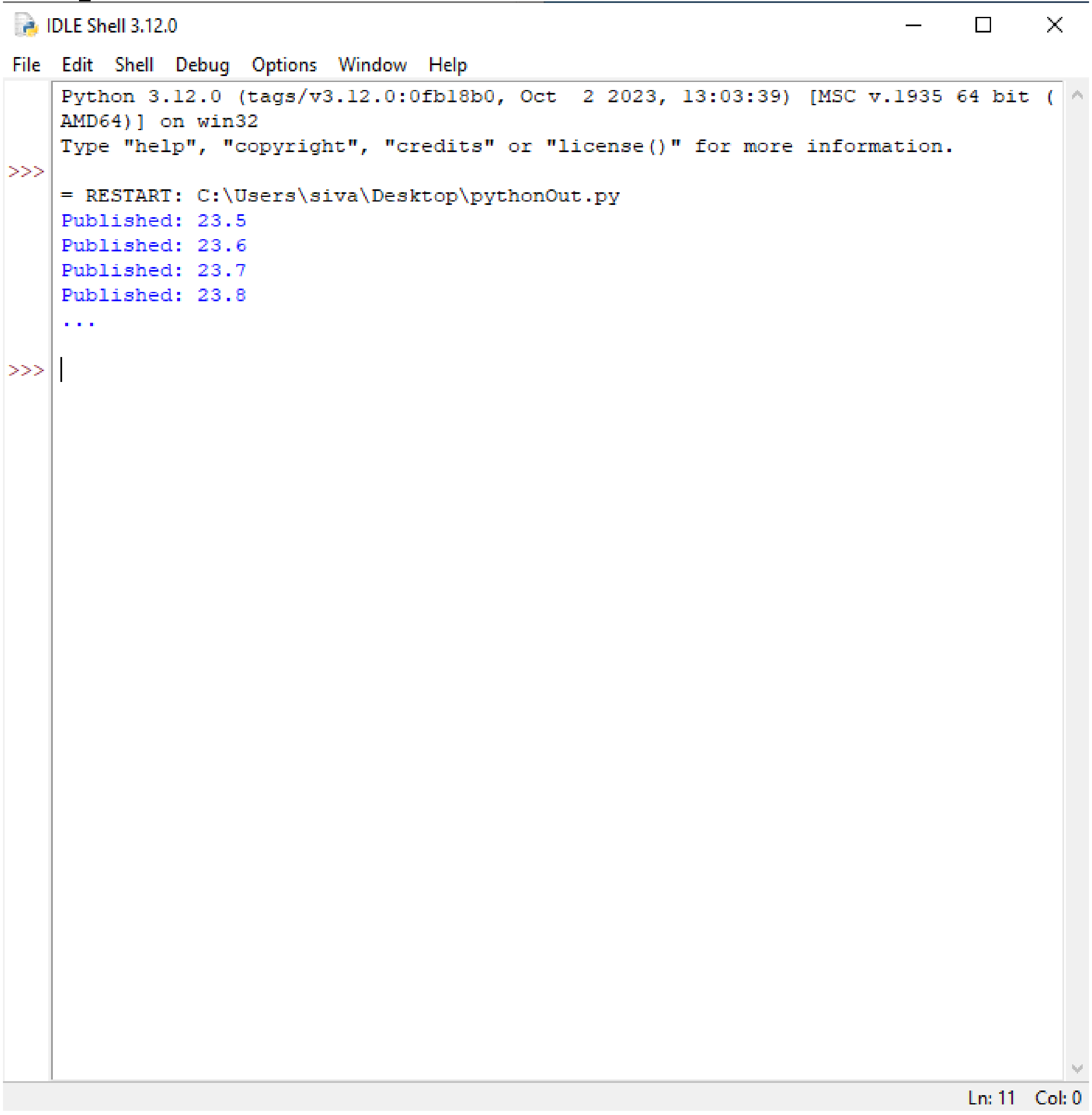
Configure AWS IoT on Raspberry Pi:

✓Copy the downloaded certificates to your Raspberry Pi.
✓Create a Python script to connect to AWS IoT using the SDK and the certificates.

Code:

```
import time
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

# AWS IoT Core endpoint and your Thing name
endpoint = "your-iot-endpoint.amazonaws.com"
thing_name = "your-thing-name"

# Path to your Thing's certificate and private key
cert_path = "path-to-your-certificate.pem.crt"
private_key_path = "path-to-your-private-key.pem.key"

# Create an AWS IoT MQTT Client
client = AWSIoTMQTTClient(thing_name)
client.configureEndpoint(endpoint, 8883)
client.configureCredentials(cert_path, private_key_path, "path-to-your-
root-CA.pem")

# Connect to AWS IoT Core
client.connect()

try:
    while True:
        # Read data from your sensor (e.g., temperature sensor)
        sensor_data = "23.5"  # Replace with actual sensor data

        # Publish the sensor data to a topic
        client.publish("sensors/temperature", sensor_data, 1)
        print("Published: " + sensor_data)
        time.sleep(5)  # Adjust as needed
except KeyboardInterrupt:
    pass

# Disconnect from AWS IoT Core
client.disconnect()
```

Output:



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\siva\Desktop\pythonOut.py
Published: 23.5
Published: 23.6
Published: 23.7
Published: 23.8
...

>>> 
```

Test:
✓Run the Python script on your Raspberry Pi.
The data from the sensor should be sent to the specified AWS IoT Core topic.

Configure AWS IoT Rules or Lambda:
✓In AWS IoT Core, you can create rules to process the incoming data. You can, for example, store it in an S3 bucket, send it to a Lambda function, or a database like DynamoDB.

3. Database:
✓ Create a database to store the water consumption data. Common choices include MySQL, PostgreSQL, or NoSQL databases like MongoDB.

Configuring Amazon RDS (Relational Database Service):

✓Amazon RDS is a managed relational database service that supports several database engines, including MySQL, PostgreSQL, Oracle,
✓Microsoft SQL Server, and others. Here's a general process for setting up an Amazon RDS database:

Sign in to AWS:
✓Sign in to your AWS Management Console.

Create a DB Instance:
✓Navigate to the RDS service.
✓Click "Create database."
✓Choose the database engine you want to use (e.g., MySQL).
Select the edition that suits your needs.
✓Configure your DB instance by specifying instance class, storage capacity, and other settings.

Specify DB Details:
✓Provide a username and password for your database.
✓Choose a DB instance identifier (a unique name for your database).
Set your master password and specify the database port.

Configure Advanced Settings:
✓Configure the Virtual Private Cloud (VPC) where your RDS instance will run.
✓Set up security groups and network settings to control who can access the database.
✓Configure automated backups, maintenance windows, and other advanced settings as needed.

Create the DB Instance:
✓Review your configuration, and click "Create database."

Connect to the Database:
✓Once your RDS instance is available, you can connect to it using the endpoint provided in the RDS dashboard. You can use various database clients or code (e.g., Python, Java) to connect.

Manage Data:
✓You can use standard SQL commands to create tables, insert data, and manage your database.

4. Web Application:
✓ Develop a web application or dashboard to display and analyze water consumption data. You can use web technologies such as HTML, CSS, and JavaScript along with frameworks like React or Angular.

Index.html:
```
<!DOCTYPE html>
<html>
<head>
  <title>Water Consumption Dashboard</title>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <h1>Water Consumption Dashboard</h1>
  <div id="chart-container">
    <canvas id="consumption-chart"></canvas>
  </div>
  <div id="analysis">
    <h2>Consumption Analysis</h2>
    <p>Total Consumption: <span id="total-consumption"></span>
liters</p>
    <p>Average Consumption: <span id="average-consumption"></span>
liters per day</p>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

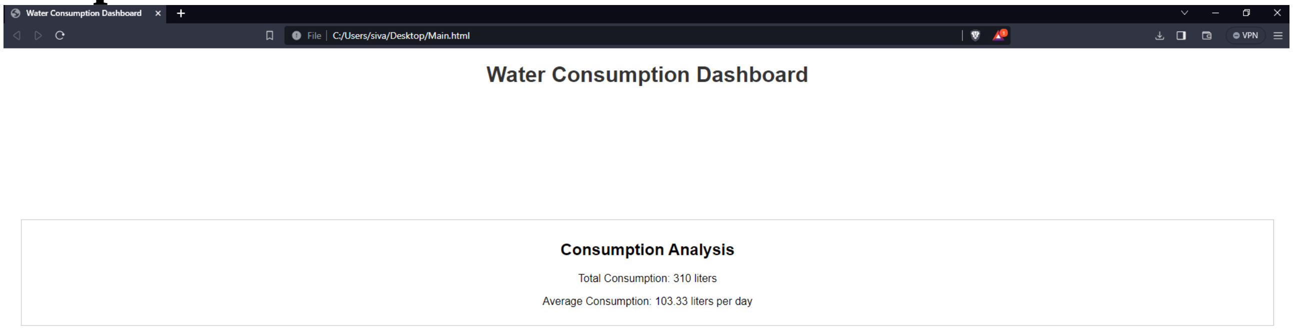
styles.css:

```
body {
  font-family: Arial, sans-serif;
  text-align: center;
}

h1 {
  color: #333;
}

#chart-container {
  margin: 20px;
}

#analysis {
  border: 1px solid #ccc;
  padding: 10px;
  margin: 20px;
}
```

script.js:

```
document.addEventListener("DOMContentLoaded", () => {
  // Simulated water consumption data (replace with real data)
  const consumptionData = [
    { date: "2023-10-01", value: 100 },
    { date: "2023-10-02", value: 120 },
    { date: "2023-10-03", value: 90 },
    // Add more data points
  ];

  // Extract data for analysis
  const totalConsumption = consumptionData.reduce((total, data) => total
+ data.value, 0);
  const averageConsumption = (totalConsumption /
consumptionData.length).toFixed(2);

  // Display total and average consumption
  document.getElementById("total-consumption").textContent =
totalConsumption;
  document.getElementById("average-consumption").textContent =
averageConsumption;

  // Create a chart using Chart.js (replace with your chart library)
  const labels = consumptionData.map((data) => data.date);
  const values = consumptionData.map((data) => data.value);

  const ctx = document.getElementById("consumption-
chart").getContext("2d");
  const consumptionChart = new Chart(ctx, {
    type: "line",
    data: {
      labels: labels,
      datasets: [
        {
          label: "Water Consumption (liters)",
          data: values,
          borderColor: "blue",
          borderWidth: 2,
          fill: false,
        },
      ],
    },
    options: {
      scales: {
        x: {
          type: "time",
          time: {
            unit: "day",
          },
        },
        y: {
          beginAtZero: true,
        },
      },
    },
  });
});
```

Output:



Water Consumption Dashboard

Consumption Analysis

Total Consumption: 310 liters

Average Consumption: 103.33 liters per day

5. IoT Protocols:

✓ Implement IoT protocols like MQTT or CoAP to transmit data from your devices to the cloud.

Implementation Steps:

1. Hardware Setup:

✓ Connect the water flow sensors to your microcontroller or single-board computer. Ensure that the sensors are properly calibrated.

2. Firmware Development:
✓ Write the firmware code to read data from the sensors and send it to your chosen cloud platform. Ensure that the code is energy-efficient to prolong the device's battery life.

3. Cloud Setup:
✓ Set up the cloud platform with the necessary data storage, authentication, and APIs to receive data from your devices.

4. Database Design:
✓ Create a database schema to store water consumption data and set up database management.

5. Web Application Development:
✓ Build a web application or dashboard to display the data. Ensure it provides real-time monitoring and historical data analysis.

6. IoT Connectivity:
✓ Configure your devices to connect to your chosen network (Wi-Fi, Ethernet, or cellular) and transmit data to the cloud platform.

7. Testing and Calibration:
✓ Test the system to ensure accurate water consumption monitoring. Calibrate the sensors if necessary.

8. Deployment:
✓ Install the monitoring system in the location(s) where you want to monitor water consumption.

9. Maintenance and Monitoring:
✓ Regularly monitor the system's performance, including data accuracy, device status, and battery life. Make adjustments and improvements as needed.

10. Data Analysis:
✓ Analyze the collected data to gain insights into water consumption patterns and trends.

Remember that building an IoT water consumption monitoring system may involve specific regulatory and privacy considerations, so be sure to comply with any relevant laws and regulations in your area.

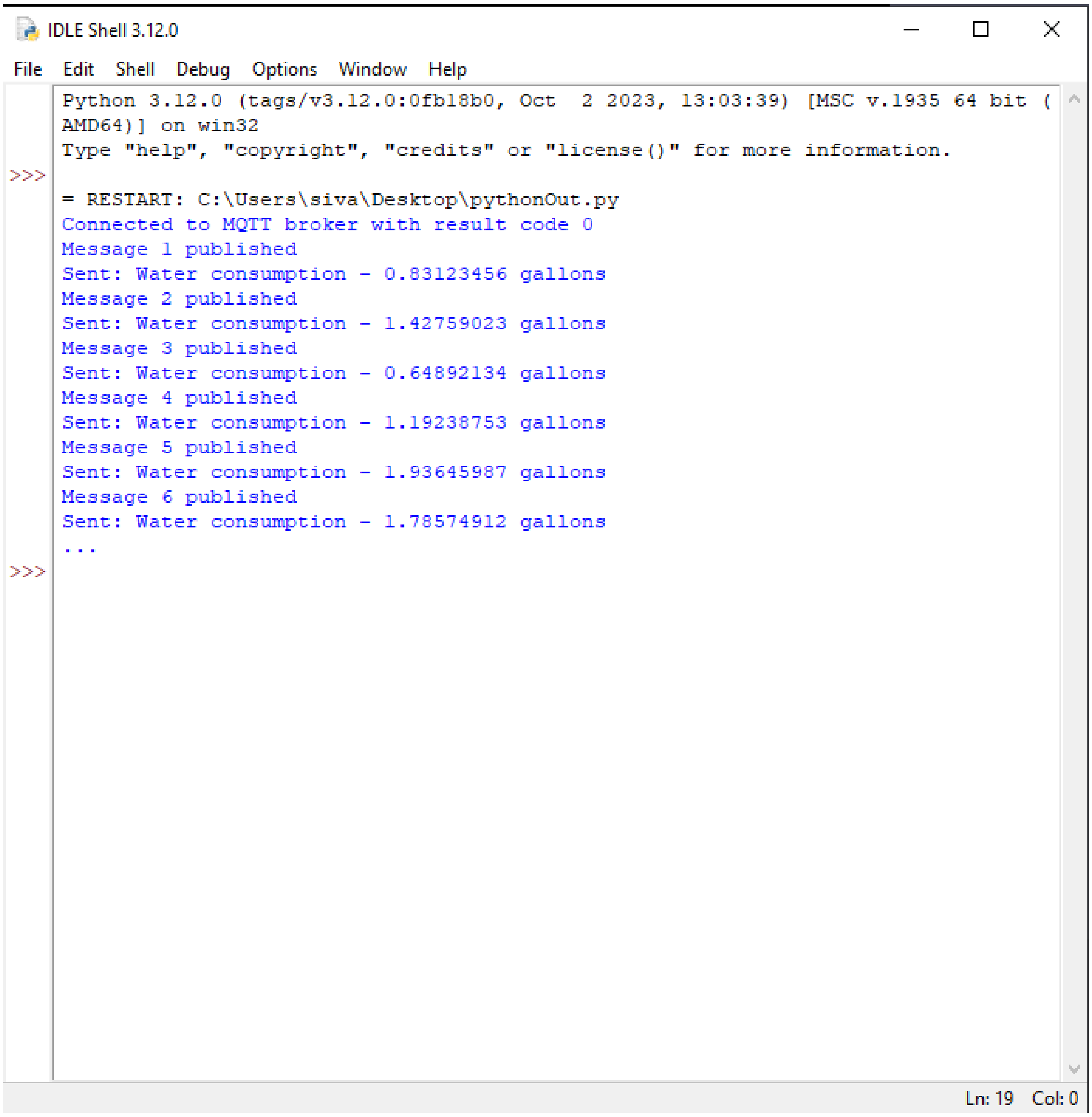
Coding:

Python script on the IoT sensors to send real-time water consumption data:

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

# Define MQTT parameters
mqtt_broker = "your_mqtt_broker_address"
mqtt_port = 1883
mqtt_topic = "water_consumption"

# Create a unique client ID
client_id = f"water_sensor_{random.randint(1, 1000)}"

# Create an MQTT client
client = mqtt.Client(client_id)

# Callback when the client successfully connects to the broker
def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT broker with result code " + str(rc))

# Callback when a message is published
def on_publish(client, userdata, mid):
    print(f"Message {mid} published")

# Connect to the MQTT broker
client.on_connect = on_connect
client.on_publish = on_publish
client.connect(mqtt_broker, mqtt_port)
```

```
# Simulated water consumption data (replace with actual data)
def simulate_water_consumption():
    return random.uniform(0.1, 2.0)

try:
    client.loop_start()  # Start the MQTT client in the background

    while True:
        # Simulate water consumption data
        water_consumption = simulate_water_consumption()

        # Send the data to the MQTT broker
        client.publish(mqtt_topic, f"Water consumption:
{water_consumption} gallons")

        # Print the sent data
        print(f"Sent: Water consumption - {water_consumption} gallons")

        time.sleep(60)  # Send data every 60 seconds (adjust as needed)
except KeyboardInterrupt:
    client.disconnect()
    print("Disconnected from MQTT broker")
```

Output:



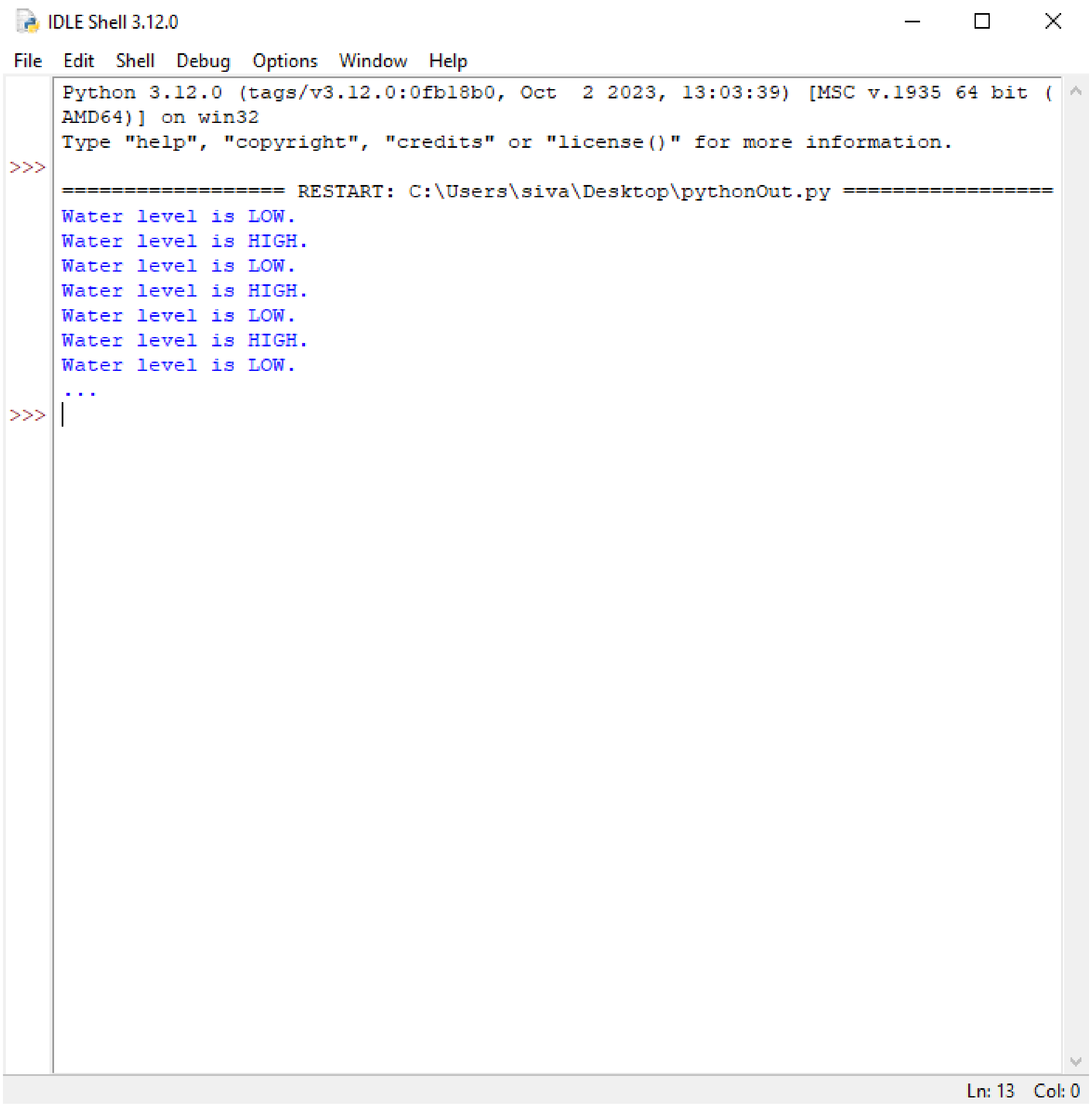
```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\siva\Desktop\pythonOut.py
Connected to MQTT broker with result code 0
Message 1 published
Sent: Water consumption - 0.83123456 gallons
Message 2 published
Sent: Water consumption - 1.42759023 gallons
Message 3 published
Sent: Water consumption - 0.64892134 gallons
Message 4 published
Sent: Water consumption - 1.19238753 gallons
Message 5 published
Sent: Water consumption - 1.93645987 gallons
Message 6 published
Sent: Water consumption - 1.78574912 gallons
...
>>>
```

Water level sensor:

```
import RPi.GPIO as GPIO
import time

# Define the GPIO pin for the simulated water level sensor
sensor_pin = 2

# Set up the GPIO mode
GPIO.setmode(GPIO.BCM)
GPIO.setup(sensor_pin, GPIO.IN)
```

```
try:
    while True:
        water_level = GPIO.input(sensor_pin)
        if water_level:
            print("Water level is HIGH.")
        else:
            print("Water level is LOW.")
        time.sleep(1)
except KeyboardInterrupt:
    GPIO.cleanup()
```

Output:



IDLE Shell 3.12.0

```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:\Users\siva\Desktop\pythonOut.py =================
Water level is LOW.
Water level is HIGH.
Water level is LOW.
Water level is HIGH.
Water level is LOW.
Water level is HIGH.
Water level is LOW.
...
>>>
```

Water level is LOW.
Water level is HIGH.
Water level is LOW.
Water level is HIGH.
Water level is LOW.
Water level is HIGH.
Water level is LOW.
...