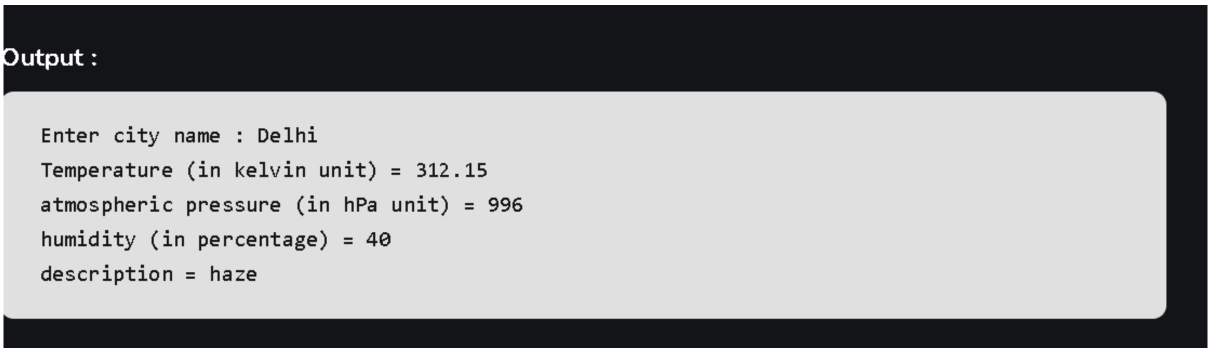
Problem 1: Real-Time Weather Monitoring System

You are developing a real-time weather monitoring system for a weather forecasting company. The system needs to fetch and display weather data for a specified location.

```
# import required modules
import requests, json

# Enter your API key here
api_key = "Your_API_Key"

# base_url variable to store url
base_url = "http://api.openweathermap.org/data/2.5/weather?"

# Give city name
city_name = input("Enter city name : ")

# complete_url variable to store
# complete url address
complete_url = base_url + "appid=" + api_key + "&q=" + city_name

# get method of requests module
# return response object
response = requests.get(complete_url)

# json method of response object
# convert json format data into
# python format data
x = response.json()

# Now x contains list of nested dictionaries
# Check the value of "cod" key is equal to
# "404", means city is found otherwise,
# city is not found
```

```
if x["cod"] != "404":

        # store the value of "main"
        # key in variable y
        y = x["main"]

        # store the value corresponding
        # to the "temp" key of y
        current_temperature = y["temp"]

        # store the value corresponding
        # to the "pressure" key of y
        current_pressure = y["pressure"]

        # store the value corresponding
        # to the "humidity" key of y
        current_humidity = y["humidity"]

        # store the value of "weather"
        # key in variable z
        z = x["weather"]

        # store the value corresponding
        # to the "description" key at
        # the 0th index of z
        weather_description = z[0]["description"]

        # print following values
        print(" Temperature (in kelvin unit) = " +
                                    str(current_temperature) +
                "\n atmospheric pressure (in hPa unit) = " +
                                    str(current_pressure) +
                "\n humidity (in percentage) = " +
                                    str(current_humidity) +
                    "\n description = " +
                                                  str(weather_description))

else:
          print(" City Not Found ")
```

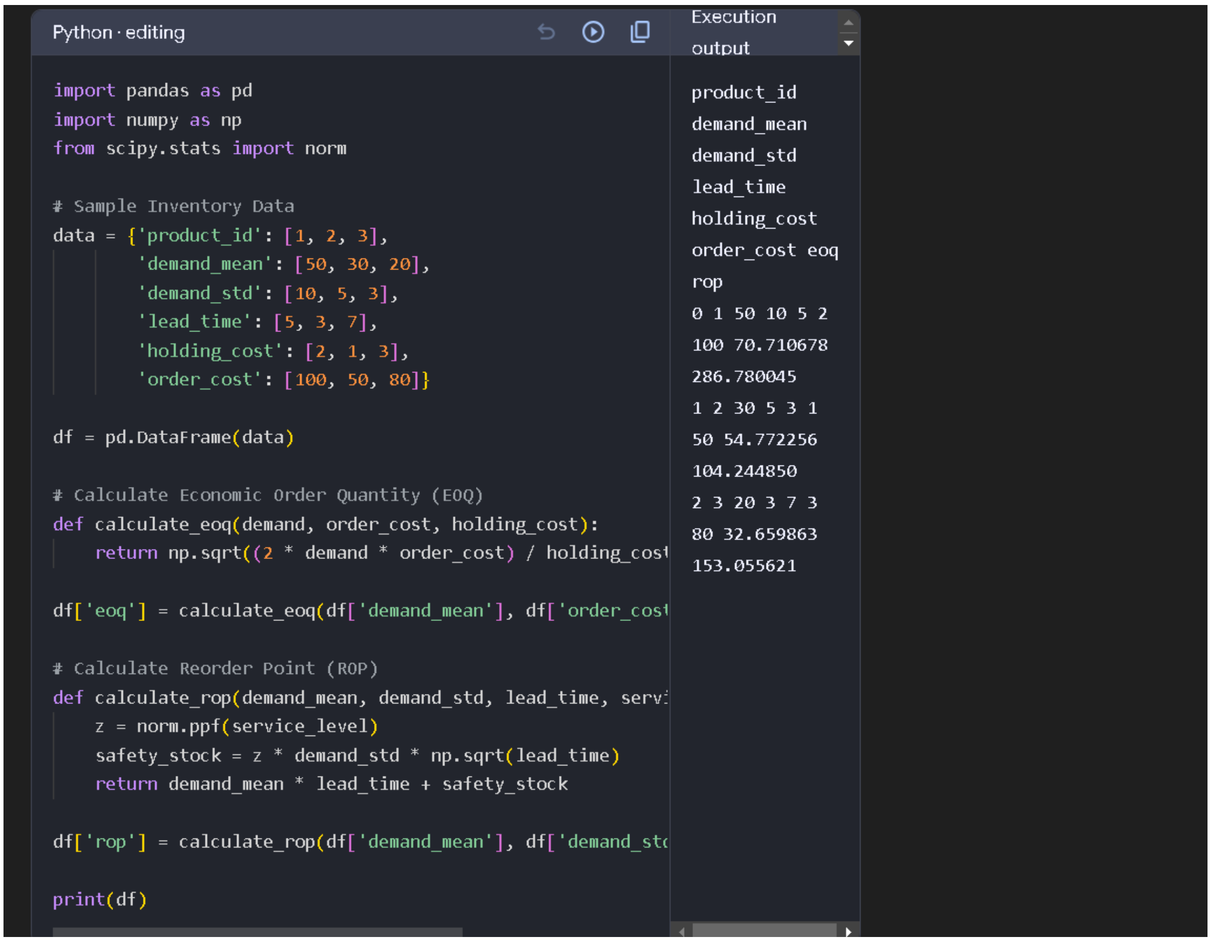


Output :

```
Enter city name : Delhi
Temperature (in kelvin unit) = 312.15
atmospheric pressure (in hPa unit) = 996
humidity (in percentage) = 40
description = haze
```

Problem 2: Inventory Management System Optimization

Scenario: You have been hired by a retail company to optimize their inventory management system. The company wants to minimize stockouts and overstock situations while maximizing inventory turnover and profitability



```python
import pandas as pd
import numpy as np
from scipy.stats import norm

# Sample Inventory Data
data = {'product_id': [1, 2, 3],
        'demand_mean': [50, 30, 20],
        'demand_std': [10, 5, 3],
        'lead_time': [5, 3, 7],
        'holding_cost': [2, 1, 3],
        'order_cost': [100, 50, 80]}

df = pd.DataFrame(data)

# Calculate Economic Order Quantity (EOQ)
def calculate_eoq(demand, order_cost, holding_cost):
    return np.sqrt((2 * demand * order_cost) / holding_cost

df['eoq'] = calculate_eoq(df['demand_mean'], df['order_cost

# Calculate Reorder Point (ROP)
def calculate_rop(demand_mean, demand_std, lead_time, servi
    z = norm.ppf(service_level)
    safety_stock = z * demand_std * np.sqrt(lead_time)
    return demand_mean * lead_time + safety_stock

df['rop'] = calculate_rop(df['demand_mean'], df['demand_stc

print(df)
```

```
product_id
demand_mean
demand_std
lead_time
holding_cost
order_cost eoq
rop
0 1 50 10 5 2
100 70.710678
286.780045
1 2 30 5 3 1
50 54.772256
104.244850
2 3 20 3 7 3
80 32.659863
153.055621
```

Problem 3: Real-Time Traffic Monitoring System

Scenario: You are working on a project to develop a real-time traffic monitoring system for a smart city initiative. The system should provide real-time traffic updates and suggest alternative routes.

```
import random
import time
import threading

# Simulating real-time traffic data
def generate_traffic_data():
    while True:
        traffic_data = {
            "location": (random.uniform(-90, 90), random.uniform(-180, 180)),
            "speed": random.uniform(0, 100),  # Speed in km/h
            "timestamp": time.time()
        }
        process_traffic_data(traffic_data)
        time.sleep(1)  # Simulate real-time data every second

def process_traffic_data(data):
    # Process and print the data (in real scenarios, this would involve more complex processing)
    print(f"Location: {data['location']}, Speed: {data['speed']} km/h, Time: 
{time.ctime(data['timestamp'])}")

# Run the simulation in a separate thread
thread = threading.Thread(target=generate_traffic_data)
thread.start()
import folium

# Create a map centered around a specific location
map_center = [0, 0]  # Centered at the equator for this example
traffic_map = folium.Map(location=map_center, zoom_start=2)
```

```
# Example of adding a marker (in real-time, you would update this dynamically)
def add_traffic_marker(location, speed):
    folium.Marker(location, popup=f"Speed: {speed} km/h").add_to(traffic_map)

# Add a sample marker
add_traffic_marker((51.5074, -0.1278), 60)  # London

# Save the map to an HTML file
traffic_map.save("traffic_map.html")
def process_traffic_data(data):
    # Process and print the data
    print(f"Location: {data['location']}, Speed: {data['speed']} km/h, Time: 
{time.ctime(data['timestamp'])}")
    # Alert if speed is below a certain threshold
    if data['speed'] < 20:
        print("Alert: Low traffic speed detected!")

# Run the simulation in a separate thread
thread = threading.Thread(target=generate_traffic_data)
thread.start()
import random
import time
import threading
import folium

# Simulating real-time traffic data
def generate_traffic_data():
    while True:
        traffic_data = {
            "location": (random.uniform(-90, 90), random.uniform(-180, 180)),
            "speed": random.uniform(0, 100),  # Speed in km/h
            "timestamp": time.time()
        }
        process_traffic_data(traffic_data)
        time.sleep(1)  # Simulate real-time data every second
```

```
def process_traffic_data(data):
    # Process and print the data
    print(f"Location: {data['location']}, Speed: {data['speed']} km/h, Time: 
{time.ctime(data['timestamp'])}")
    # Alert if speed is below a certain threshold
    if data['speed'] < 20:
        print("Alert: Low traffic speed detected!")
    # Add to map
    add_traffic_marker(data['location'], data['speed'])


def add_traffic_marker(location, speed):
    folium.Marker(location, popup=f"Speed: {speed} km/h").add_to(traffic_map)


# Create a map centered around a specific location
map_center = [0, 0]  # Centered at the equator for this example
traffic_map = folium.Map(location=map_center, zoom_start=2)


# Run the simulation in a separate thread
thread = threading.Thread(target=generate_traffic_data)
thread.start()


# Save the map periodically to reflect real-time changes
def save_map_periodically():
    while True:
        traffic_map.save("traffic_map.html")
        time.sleep(10)  # Update the map every 10 seconds


save_thread = threading.Thread(target=save_map_periodically)
save_thread.start()
```

Problem 4: Real-Time COVID-19 Statistics Tracker

Scenario: You are developing a real-time COVID-19 statistics tracking application for a healthcare organization. The application should provide up-to-date information on COVID-19 cases, recoveries, and deaths for a specified region

(pip install requests folium)

```
import requests

import time

import threading

import folium


# Function to fetch COVID-19 data from the API

def fetch_covid_data():

    url = "https://disease.sh/v3/covid-19/countries"

    response = requests.get(url)

    return response.json()


# Function to process and add COVID-19 data to the map

def process_covid_data(data):

    global covid_map

    for country in data:

        location = [country['countryInfo']['lat'], country['countryInfo']['long']]

        cases = country['cases']

        deaths = country['deaths']

        recovered = country['recovered']

        folium.Marker(location,

                      popup=f"Country: {country['country']}<br>Cases: {cases}<br>Deaths:
{deaths}<br>Recovered: {recovered}").add_to(covid_map)


# Function to update the COVID-19 map

def update_covid_map():

    global covid_map

    while True:

        data = fetch_covid_data()

        covid_map = folium.Map(location=[0, 0], zoom_start=2)  # Reset map

        process_covid_data(data)
        covid_map.save("covid_map.html")
        time.sleep(3600)  # Update every hour

# Initialize the map
covid_map = folium.Map(location=[0, 0], zoom_start=2)

# Start the map update in a separate thread
thread = threading.Thread(target=update_covid_map)
thread.start()
```