PART 5 - Script

🧠 SECTION 1: WHAT IS AN API? (Explain BEFORE Code)

"First, let's understand what an API is.
API stands for Application Programming Interface.
It allows one application to communicate with another application."

"Here, our Python script is a **client**, and the weather service is a **server**."

"We send a request, and the server sends a response."

SECTION 2: WHAT IS HTTP & GET REQUEST?

"HTTP is a communication protocol used on the internet.
There are different types of HTTP methods like GET, POST, PUT, DELETE."

"GET is used when we want to **fetch data**.
In our case, we are fetching weather data, so we use a GET request."

📦 SECTION 3: WHAT IS JSON?

"The server sends data in JSON format.
JSON stands for JavaScript Object Notation."

"JSON looks like a Python dictionary and is very easy to convert into Python data structures."

"Example: temperature, humidity, wind speed are all sent inside JSON."

🔐 SECTION 4: API KEY & HEADERS (IMPORTANT)

"Most real APIs require authentication.
This is done using an API key."

"An API key identifies who is making the request."

"We never hardcode API keys.
We store them in environment variables using a .env file."

🧩 SECTION 5: REQUEST vs RESPONSE FLOW

"So the full flow is:
User gives input → Python builds request → API server processes it → JSON response comes back → Python extracts data → Output is shown."

🟢 IMPORTS

```
from dotenv import load_dotenv

import requests

import os
```

🎙 **Speak this:**

"Here, dotenv is used to load environment variables.
requests is used to send HTTP requests.
os helps us access system environment variables."

🟢 LOAD ENVIRONMENT VARIABLES

```
load_dotenv()
```

🎙 **Say:**

"This loads the API key from the .env file into the program memory."

🟢 API CONFIGURATION

```
BASE_URL = "http://api.openweathermap.org/data/2.5/weather"

API_KEY = os.getenv("API_KEY")
```

🎙 **Say:**

"This is the API endpoint where we send our request.
The API key is fetched securely using os.getenv."

🟢 FUNCTION DEFINITION

```
def get_weather(city_name: str):
```

🎙 **Say:**

"This function takes a city name and returns weather data or an error message."

🟢 QUERY PARAMETERS

```
params = {
  "q": city_name,
  "appid": API_KEY,
  "units": "metric"
}
```

🎙 **Explain clearly:**

“These are query parameters.
q is the city name,
appid is the API key,
units=metric means temperature will be in Celsius.”

🟢 TRY BLOCK – NETWORK REQUEST

```
try:
  response = requests.get(BASE_URL, params=params, timeout=10)
```

🎙 **Say slowly:**

“Here, we send the HTTP GET request.
timeout=10 means the request will wait for 10 seconds maximum.”

🔴 EXCEPT BLOCK – NETWORK ERRORS

```
except requests.exceptions.RequestException as e:
  return {"error": f"Network error: {e}"}
```

🎙 **Say:**

“This block handles network-related issues like no internet, DNS failure, or timeout.
Instead of crashing, we return a clean error message.”

🟠 HTTP STATUS CODE HANDLING

```
if response.status_code != 200:
```

🎙 **Explain:**

“HTTP status code 200 means success.
Any other code means something went wrong.”

Parse Error JSON

```
try:
    error_data = response.json()
    message = error_data.get("message", "Unknown error")
except ValueError:
    message = "Invalid server response"
```

🎙 **Say:**

“Sometimes the server sends error details in JSON.
We safely extract the message without crashing.”

Handle Specific Errors

```
if response.status_code == 401:
```

🎙

“401 means unauthorized — invalid API key.”

```
elif response.status_code == 404:
```

🎙

“404 means city not found.”

```
else:
```

🎙

“Any other status code is handled generically.”

● **PARSE SUCCESS JSON**

```
data = response.json()
```

🎙 **Say:**

“Here we convert JSON response into a Python dictionary.”

🟢 WHY WE EXTRACT DATA LIKE THIS (IMPORTANT INTERVIEW POINT)

```
main = data.get("main", {})

weather = data.get("weather", [])

wind = data.get("wind", {})
```

🎙 VERY IMPORTANT — Say this clearly:

"We use .get() instead of direct indexing to avoid KeyError.
If data is missing, the program will not crash."

"This is a **production-level best practice**."

🟢 FINAL CLEAN OUTPUT

```
return {

  "city": data.get("name", city_name),

  "temperature": main.get("temp"),

  "humidity": main.get("humidity"),

  "pressure": main.get("pressure"),

  "condition": weather[0].get("description") if weather else None,

  "wind_speed": wind.get("speed")

}
```

🎙 Say:

"We return a clean dictionary that contains only useful data.
This makes the output reusable for CLI, UI, or backend services."

🟢 MAIN EXECUTION BLOCK

```
if __name__ == "__main__":
```

🎙 Explain:

"This ensures the code runs only when the file is executed directly."

🟢 USER INPUT

city = input("Enter city name: ").strip()

🎙

“We take user input and remove extra spaces.”

🟢 FINAL OUTPUT

weather_data = get_weather(city)

print_weather_report(weather_data)

🎙

“Finally, we fetch the weather data and print it in a clean format.”

🎯 CLOSING STATEMENT (IMPORTANT)

“In this video, we learned APIs, HTTP requests, JSON handling, error handling, and production-ready Python automation.
In the next part, we will cover POST requests, headers, authentication tokens, and real backend workflows.”

```
# -------------------------------------------------------
# Python Weather CLI Tool using OpenWeatherMap API
# -------------------------------------------------------
# WHAT THIS SCRIPT DOES:
# 1. Takes a city name from the user
# 2. Sends an HTTP GET request to OpenWeatherMap API
# 3. Receives weather data in JSON format
# 4. Safely extracts required fields
# 5. Handles all possible errors
# 6. Displays a clean weather report in terminal
# -------------------------------------------------------


# -------------------------------------------------------
# IMPORT SECTION
# -------------------------------------------------------

# load_dotenv is used to load environment variables
# from a .env file into the system environment
from dotenv import load_dotenv
```

```
# requests is a third-party library used to send HTTP requests
import requests

# os is used to access environment variables securely
import os


# --------------------------------------------------------
# CONFIGURATION / CONSTANTS
# --------------------------------------------------------

# Load environment variables from .env file
# This allows us to keep API keys outside the code
load_dotenv()

# Base URL (API Endpoint)
# This is the server address where the request is sent
BASE_URL = "http://api.openweathermap.org/data/2.5/weather"

# Fetch API key securely from environment variables
# This avoids hard-coding sensitive information
API_KEY = os.getenv("API_KEY")


# --------------------------------------------------------
# FUNCTION: FETCH WEATHER DATA
# --------------------------------------------------------
def get_weather(city_name: str):
    """
    Fetch weather information for a given city.

    Parameters:
        city_name (str): Name of the city entered by user

    Returns:
        dict: Weather details OR error information
    """

    # --------------------------------------------------------
    # STEP 1: BUILD QUERY PARAMETERS
    # --------------------------------------------------------
    # These parameters are attached to the URL
    # ?q=Pune&appid=XXXX&units=metric
    params = {
        "q": city_name,      # City name
        "appid": API_KEY,    # API authentication key
        "units": "metric"    # Metric units → Celsius
    }


    # --------------------------------------------------------
    # STEP 2: SEND HTTP GET REQUEST
    # --------------------------------------------------------
    # try block is used because network calls can fail
    try:
        response = requests.get(
            BASE_URL,        # API endpoint
            params=params,   # Query parameters
            timeout=10       # Max wait time (seconds)
        )

    # This catches ALL request-related errors:
    # - No internet
    # - Timeout
    # - DNS failure
    # - Connection error
    except requests.exceptions.RequestException as e:
        return {
            "error": f"Network error occurred: {e}"
        }


    # --------------------------------------------------------
    # STEP 3: CHECK HTTP STATUS CODE
    # --------------------------------------------------------
    # HTTP 200 means success
    if response.status_code != 200:

        # Try reading error message sent by server
        try:
            error_data = response.json()
            message = error_data.get("message", "Unknown error")
        except ValueError:
            # Happens if server response is not valid JSON
            message = "Invalid response from server"

        # Handle common HTTP errors
        if response.status_code == 401:
            return {"error": "Invalid API key"}
        elif response.status_code == 404:
            return {"error": f"City not found: {city_name}"}
        else:
            return {
                "error": f"HTTP {response.status_code}: {message}"
            }


    # --------------------------------------------------------
    # STEP 4: PARSE JSON RESPONSE
    # --------------------------------------------------------
    # Convert JSON → Python dictionary
    try:
        data = response.json()
    except ValueError:
        return {"error": "Failed to parse JSON response"}


    # --------------------------------------------------------
    # STEP 5: SAFE DATA EXTRACTION (IMPORTANT CONCEPT)
    # --------------------------------------------------------
    # WHY WE DO THIS:
    # - API responses can change
    # - Keys may be missing
    # - Accessing missing keys directly causes crashes

    # Extract "main" section safely
    # Example: data["main"] → {"temp": 30, "humidity": 60}
    main = data.get("main", {})

    # Extract "weather" list safely
    # Example: [{"description": "clear sky"}]
    weather = data.get("weather", [])

    # Extract "wind" section safely
    # Example: {"speed": 5.6}
    wind = data.get("wind", {})


    # --------------------------------------------------------
    # STEP 6: BUILD CLEAN RESULT DICTIONARY
    # --------------------------------------------------------
    return {
        # City name returned by API
        "city": data.get("name", city_name),

        # Temperature in Celsius
        "temperature": main.get("temp"),

        # Humidity percentage
        "humidity": main.get("humidity"),
        # Atmospheric pressure
        "pressure": main.get("pressure"),

        # Weather condition (first item from list)
        "condition": weather[0].get("description") if weather else None,

        # Wind speed
        "wind_speed": wind.get("speed")
    }


# -------------------------------------------------------
# FUNCTION: DISPLAY WEATHER REPORT
# -------------------------------------------------------
def print_weather_report(result: dict):
    """
    Prints formatted weather report or error message
    """

    # If API returned an error
    if "error" in result:
        print("\n❌ Error:", result["error"])
        return

    # Display clean formatted output
    print("\n📍 Weather Report for:", result["city"])
    print("----------------------------------------")
    print(f"🌡 Temperature : {result['temperature']} °C")
    print(f"💧 Humidity    : {result['humidity']} %")
    print(f"🔽 Pressure    : {result['pressure']} hPa")
    print(f"🌬 Wind Speed  : {result['wind_speed']} m/s")
    print(f"🌦 Condition   : {result['condition']}")


# -------------------------------------------------------
# PROGRAM ENTRY POINT
# -------------------------------------------------------
if __name__ == "__main__":

    # Take city name from user
    city = input("Enter city name: ").strip()

    # Validate input
    if not city:
        print("Please enter a valid city name.")
```

```
else:
   # Fetch weather data
   weather_data = get_weather(city)

   # Print result
   print_weather_report(weather_data)
```