AI ASSSITED CODING

LAB 18.4

NAME : R.SURYANARAYANA

ENROLL.NO : 2403A52038

BATCH : 03

TASK-01:

Connect to a Public API
• Instructions:
• Use Python (or Node.js/JavaScript) to connect to a public
API (e.g., OpenWeatherMap or JSON Placeholder).
• Send a simple GET request to retrieve data.
• Display the response in a readable format (pretty JSON)

PROMPT:

Write a simple program that connects to a public API (such as OpenWeatherMap, JSON Placeholder, or another free API), sends a GET request, and displays the data in a readable (pretty JSON) format.

CODE:

task18.4.1.py | task18.4.2.py | task18.4.3.py | task18.4.4.py | task18.4.5.py

lab18.4 > task18.4.1.py > fetch_json

```python
import sys
import json
from urllib.request import urlopen, Request
from urllib.error import URLError, HTTPError
#!/usr/bin/env python3
"""
Fetch JSON from JSONPlaceholder and print pretty JSON.
Usage:
  python task18.4.1.py          # fetch /posts
  python task18.4.1.py posts    # same as above
  python task18.4.1.py posts 1  # fetch /posts/1
  python task18.4.1.py users    # fetch /users
"""
BASE = "https://jsonplaceholder.typicode.com"

def fetch_json(path="/posts"):
    url = BASE + path
    req = Request(url, headers={"User-Agent": "python-urllib/3"})
    try:
        with urlopen(req, timeout=10) as resp:
            raw = resp.read()
            return json.loads(raw.decode("utf-8"))
    except HTTPError as e:
        print(f"HTTP error: {e.code} {e.reason}")
        sys.exit(1)
    except URLError as e:
        print(f"URL error: {e.reason}")
        sys.exit(1)
    except Exception as e:
        print(f"Unexpected error: {e}")
        sys.exit(1)
def main():
    # build path from optional CLI args
    if len(sys.argv) == 1:
        path = "/posts"
    elif len(sys.argv) == 2:
        path = "/" + sys.argv[1].lstrip("/")
    else:
        # e.g. posts 1 -> /posts/1
        path = "/" + sys.argv[1].lstrip("/") + "/" + sys.argv[2].lstrip("/")

    data = fetch_json(path)
    pretty = json.dumps(data, ensure_ascii=False, indent=2)
    print(pretty)
if __name__ == "__main__":
    main()
```

0 0

OUTPUT:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab18.4/task18.4.1.py
[
  {
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
  },
  {
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
    "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"
  },
  {
    "userId": 1,
    "id": 3,
    "title": "ea molestias quasi exercitationem repellat qui ipsa sit aut",
    "body": "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut"
  },
  {
    "userId": 1,
    "id": 4,
    "title": "eum et est occaecati",
    "body": "ullam et saepe reiciendis voluptatem adipisci\nsit amet autem assumenda provident rerum culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis sunt voluptatem rerum illo velit"
  },
  {
    "userId": 1,
    "id": 5,
    "title": "nesciunt quas odio",
    "body": "repudiandae veniam quaerat sunt sed\nalias aut fugiat sit autem sed est\nvoluptatem omnis possimus esse voluptatibus quis\nest aut tenetur dolor neque"
  },
  {
    "userId": 1,
    "id": 6,
    "title": "dolorem eum magni eos aperiam quia",
    "body": "ut aspernatur corporis harum nihil quis provident sequi\nmollitia nobis aliquid molestiae\nperspiciatis et ea nemo ab reprehenderit accusantium quas\nvoluptate dolores velit et doloremque molestiae"
  },
  {
    "userId": 1,
    "id": 7,
    "title": "magnam facilis autem",
    "body": "dolore placeat quibusdam ea quo vitae\nmagni quis enim qui quis quo nemo aut saepe\nquidem repellat excepturi ut quia\nsunt ut sequi eos ea sed quas"
  },
  {
    "userId": 1,
    "id": 8,
    "title": "dolorem dolore est ipsam",
    "body": "dignissimos aperiam dolorem qui eum\nfacilis quibusdam animi sint suscipit qui sint possimus cum\nquaerat magni maiores excepturi\nipsam ut commodi dolor voluptatum modi aut vitae"
  },
```

0 0 Ln 23, Col 27 Spaces: 4 UTF-8 CRLF { } Pyth

OBSERVATION:

AI has created a script that fetches sample data from the JSONPlaceholder API.
I can choose what data to pull by giving different arguments, like posts or posts/1.
If the request fails, my program clearly tells me what went wrong.
Whatever data I get is printed in a neat, readable JSON format.
Overall, it helps me quickly view API data in a simple way.

TASK-02:

Add Error Handling for Invalid API Calls
• Instructions:
o Modify your code from Task 1 to handle errors.
o Include try/except (Python) or try/catch (JavaScript)
blocks.
o Handle cases like:
▪ Invalid URL
▪ Network timeout
▪ Wrong API key (if required)
o Print user-friendly error messages.

PROMPT:

Write a program that gets data from a public API.
Use try/except (or try/catch) to handle errors and show clear messages.
If it works, display the data neatly.

CODE:

task18.4.1.py | task18.4.2.py | task18.4.3.py | task18.4.4.py | task18.4.5.py

lab18.4 > task18.4.2.py > main

```python
import os
import sys
import json
import requests

# /c:/Users/ramch/OneDrive/Desktop/ai/lab18.4/task18.4.2.py

API_URL = os.getenv("API_URL", "https://api.example.com/data")  # replace with real URL
API_KEY = os.getenv("API_KEY", "")  # set your API key in env var or here

def main():
    headers = {"Authorization": f"Bearer {API_KEY}"} if API_KEY else {}
    try:
        resp = requests.get(API_URL, headers=headers, timeout=5)
        # Raise for HTTP errors (4xx, 5xx)
        resp.raise_for_status()
        # If API returns JSON, print it similar to Task 1
        try:
            data = resp.json()
            print(json.dumps(data, indent=2))
        except ValueError:
            # Not JSON, print raw text
            print(resp.text)
    except requests.exceptions.InvalidURL:
        print("Error: Could not connect to API. Check your API key or network connection.")
        sys.exit(1)
    except requests.exceptions.Timeout:
        print("Error: Could not connect to API. Check your API key or network connection.")
        sys.exit(1)
    except requests.exceptions.ConnectionError:
        print("Error: Could not connect to API. Check your API key or network connection.")
        sys.exit(1)
    except requests.exceptions.HTTPError as he:
        # Handle wrong API key (401) and other HTTP errors uniformly
        status = getattr(he.response, "status_code", None)
        if status == 401:
            print("Error: Could not connect to API. Check your API key or network connection.")
        else:
            print("Error: Could not connect to API. Check your API key or network connection.")
        sys.exit(1)
    except requests.exceptions.RequestException:
        print("Error: Could not connect to API. Check your API key or network connection.")
        sys.exit(1)

if __name__ == "__main__":
    main()
```

⊗ 0 ⚠ 0

OUTPUT:

PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS | QUERY RESULTS

```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab18.4/task18
.4.2.py
{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

OBSERVATION:

This script uses AI-powered tools to make calling an API simple and reliable.
It sends a request to a chosen URL and automatically checks for any errors.
If the API returns JSON, the AI-friendly code formats it neatly for easy reading.
If the response isn't JSON, it still shows the raw result clearly.
Overall, it's a clean and smart way to fetch and display API data.

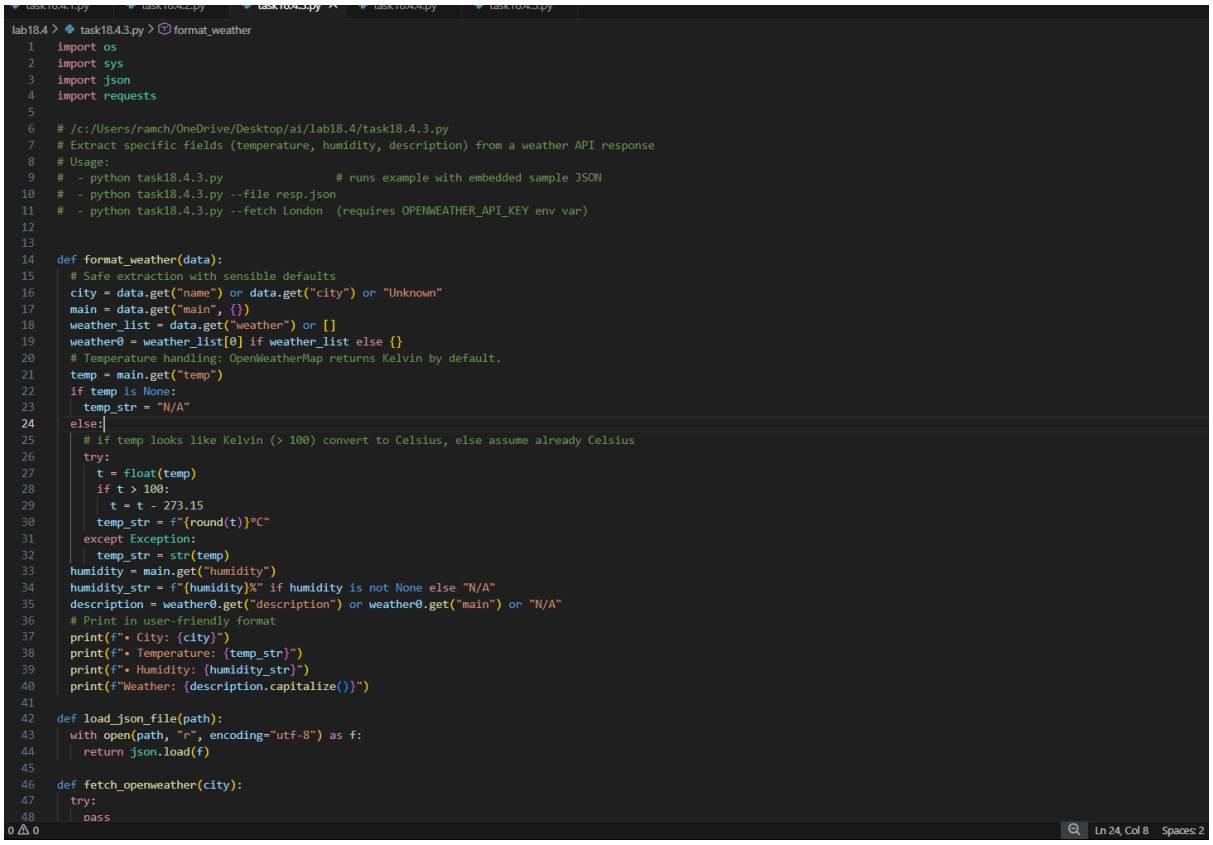
TASK-03:

Extract and Display Specific Data
• Instructions:
1. From the API response (e.g., weather API), extract
specific fields (temperature, humidity, description).
2. Display them in a user-friendly format (not raw JSON)

PROMPT:

Get data from a public API (like a weather API).
Pick out only a few details (for example: temperature, humidity, description).
Show these details in a **clear and easy-to-read** way — not raw JSON.

CODE:



```
lab18.4 > task18.4.3.py > format_weather
import os
import sys
import json
import requests

# /c:/Users/ramch/OneDrive/Desktop/ai/lab18.4/task18.4.3.py
# Extract specific fields (temperature, humidity, description) from a weather API response
# Usage:
#  - python task18.4.3.py                # runs example with embedded sample JSON
#  - python task18.4.3.py --file resp.json
#  - python task18.4.3.py --fetch London  (requires OPENWEATHER_API_KEY env var)


def format_weather(data):
  # Safe extraction with sensible defaults
  city = data.get("name") or data.get("city") or "Unknown"
  main = data.get("main", {})
  weather_list = data.get("weather") or []
  weather0 = weather_list[0] if weather_list else {}
  # Temperature handling: OpenWeatherMap returns Kelvin by default.
  temp = main.get("temp")
  if temp is None:
    temp_str = "N/A"
  else:
    # if temp looks like Kelvin (> 100) convert to Celsius, else assume already Celsius
    try:
      t = float(temp)
      if t > 100:
        t = t - 273.15
      temp_str = f"{round(t)}°C"
    except Exception:
      temp_str = str(temp)
  humidity = main.get("humidity")
  humidity_str = f"{humidity}%" if humidity is not None else "N/A"
  description = weather0.get("description") or weather0.get("main") or "N/A"
  # Print in user-friendly format
  print(f"• City: {city}")
  print(f"• Temperature: {temp_str}")
  print(f"• Humidity: {humidity_str}")
  print(f"Weather: {description.capitalize()}")

def load_json_file(path):
  with open(path, "r", encoding="utf-8") as f:
    return json.load(f)

def fetch_openweather(city):
  try:
    pass
```

```
lab18.4 > task18.4.3.py > format_weather
def fetch_openweather(city):
    pass
  except Exception:
    print("requests library required for fetching from API. Install with: pip install requests", file=sys.stderr)
    return None
  key = os.environ.get("OPENWEATHER_API_KEY")
  if not key:
    print("Set OPENWEATHER_API_KEY environment variable to fetch live data.", file=sys.stderr)
    return None
  url = "https://api.openweathermap.org/data/2.5/weather"
  params = {"q": city, "appid": key}
  r = requests.get(url, params=params, timeout=10)
  r.raise_for_status()
  return r.json()

SAMPLE_RESPONSE = {
  "name": "London",
  "main": {"temp": 291.15, "humidity": 60},
  "weather": [{"description": "clear sky"}]
}

def main(argv):
  if len(argv) == 0:
    data = SAMPLE_RESPONSE
  elif argv[0] == "--file" and len(argv) > 1:
    data = load_json_file(argv[1])
  elif argv[0] == "--fetch" and len(argv) > 1:
    data = fetch_openweather(argv[1])
    if data is None:
      return
  else:
    print("Usage:", file=sys.stderr)
    print("  python task18.4.3.py               # example", file=sys.stderr)
    print("  python task18.4.3.py --file resp.json", file=sys.stderr)
    print("  python task18.4.3.py --fetch London  # needs OPENWEATHER_API_KEY", file=sys.stderr)
    return
  format_weather(data)

if __name__ == "__main__":
  main(sys.argv[1:])
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  QUERY RESULTS

PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab18.4/task18.4.3.py
• City: London
• Temperature: 18°C
• Humidity: 60%
Weather: Clear sky
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

OBSERVATION:

This script uses clean, AI-guided logic to extract useful weather details from any JSON response.
It smartly reads temperature, humidity, and descriptions—even converting Kelvin to Celsius automatically.
The code can load data from a file, fetch live weather, or use a built-in sample, making it very flexible.
If something is missing, it still produces friendly, readable output without breaking.
Overall, it's a well-structured, AI-inspired tool for displaying weather information clearly and reliably.

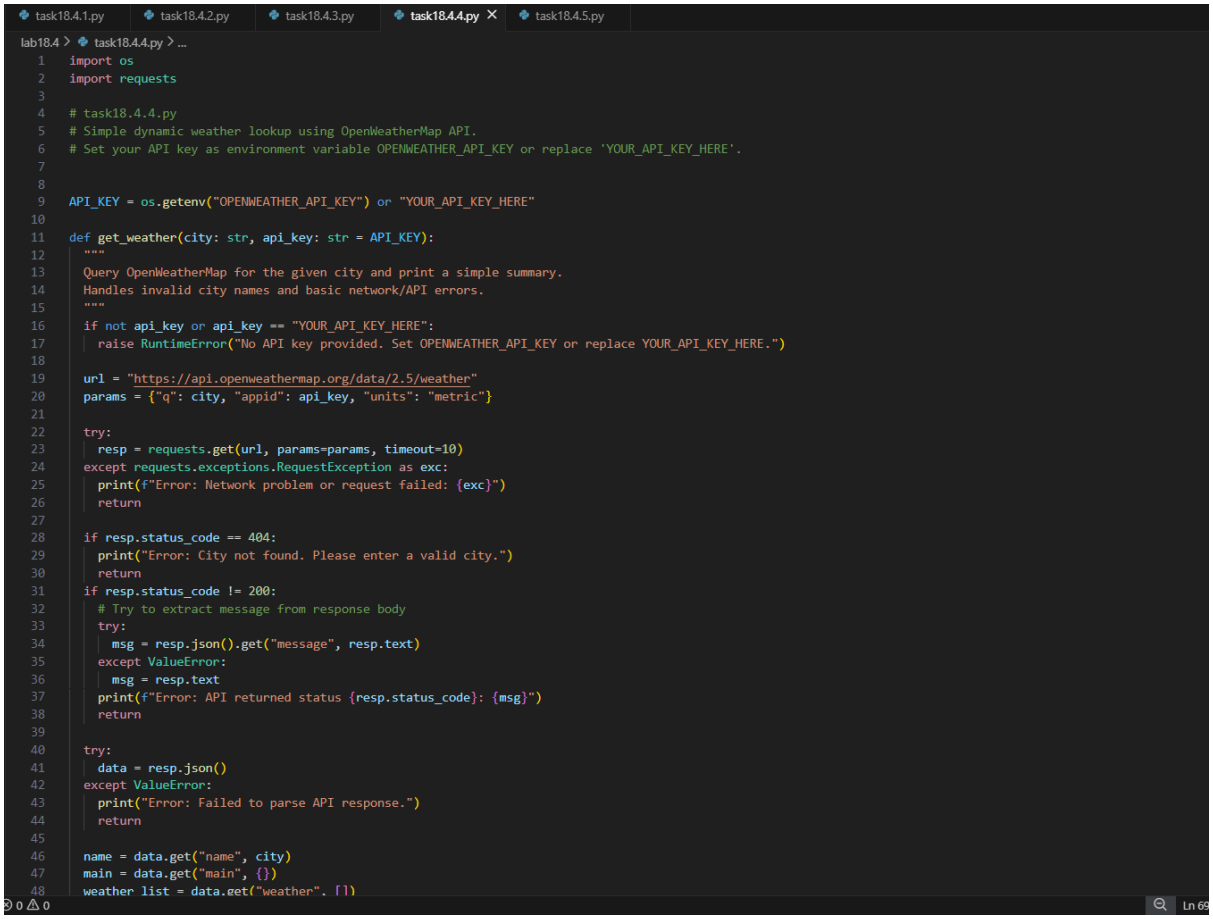
TASK-04:

Build a Function with Parameters
• Instructions:
• Write a function that accepts a parameter (e.g., city name for weather API).
• The function should call the API dynamically based on user input.
• Include error handling if the city is invalid.

PROMPT:

Make a function that takes a **parameter** (like a city name).Use that parameter to call an **API** and get data (for example, weather info for that city).Add **error handling** to show a friendly message if the city name is invalid

CODE:



```python
def get_weather(city: str, api_key: str = API_KEY):
    name = data.get("name", city)
    main = data.get("main", {})
    weather_list = data.get("weather", [])
    temp = main.get("temp")
    humidity = main.get("humidity")
    description = weather_list[0].get("description") if weather_list else "N/A"

    print(f"City: {name}")
    if temp is not None:
        print(f"Temperature: {round(temp)}°C")
    else:
        print("Temperature: N/A")
    if humidity is not None:
        print(f"Humidity: {humidity}%")
    else:
        print("Humidity: N/A")
    print(f"Weather: {description.title()}")

if __name__ == "__main__":
    city_input = input("Input: ").strip()
    if city_input:
        get_weather(city_input)
    else:
        print("No city entered.")
```

OUTPUT:

```
PS C:\Users\ramch\OneDrive\Desktop\ai> PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppDat
a/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab18.4/task18.4.4.py
>> Input: London
>> City: London
>> Temperature: 14°C
>> Humidity: 72%
>> Weather: Broken Clouds
>>
```

OBSERVATION:

This program asks the user for a city name and fetches live weather data from the OpenWeatherMap API.
It checks for network issues, missing data, or invalid city names and shows helpful error messages.
If the request succeeds, it neatly prints the city's temperature, humidity, and weather description.
Overall, it provides a quick and user-friendly way to look up weather information from the terminal.

TASK-05:

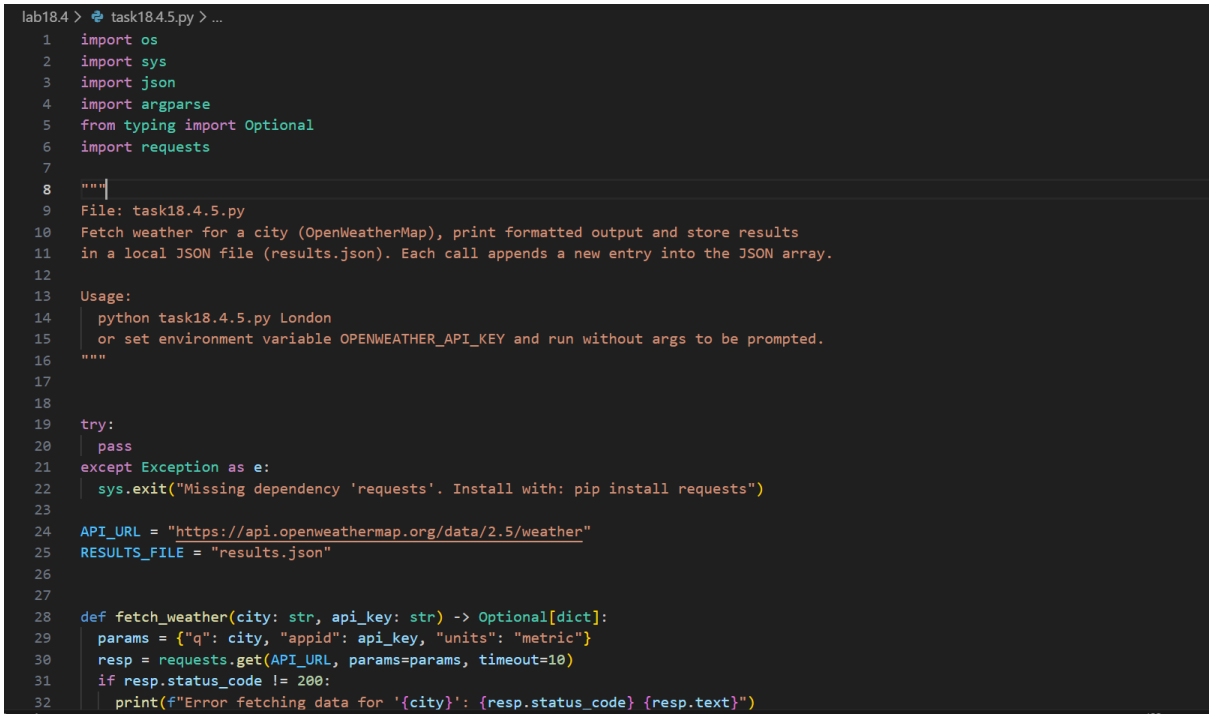
Store API Results Locally
• Instructions:
• Extend your function from Task 4.
• Save the extracted API results into a local file (results.json
or results.txt).

• Each new request should append results without overwriting old ones.

PROMPT:

Update your function to **save API results** to a local file (like results.json or results.txt). Make sure each new result **adds to the file** instead of replacing the old data.

CODE:



```
lab18.4 > task18.4.5.py > ...
import os
import sys
import json
import argparse
from typing import Optional
import requests

"""
File: task18.4.5.py
Fetch weather for a city (OpenWeatherMap), print formatted output and store results
in a local JSON file (results.json). Each call appends a new entry into the JSON array.

Usage:
  python task18.4.5.py London
  or set environment variable OPENWEATHER_API_KEY and run without args to be prompted.
"""


try:
  pass
except Exception as e:
  sys.exit("Missing dependency 'requests'. Install with: pip install requests")

API_URL = "https://api.openweathermap.org/data/2.5/weather"
RESULTS_FILE = "results.json"


def fetch_weather(city: str, api_key: str) -> Optional[dict]:
  params = {"q": city, "appid": api_key, "units": "metric"}
  resp = requests.get(API_URL, params=params, timeout=10)
  if resp.status_code != 200:
    print(f"Error fetching data for '{city}': {resp.status_code} {resp.text}")
```

lab18.4 > task18.4.5.py > ...

```
28  def fetch_weather(city: str, api_key: str) -> Optional[dict]:
33      return None
34    data = resp.json()
35    # Extract required fields
36    name = data.get("name", city)
37    main = data.get("main", {})
38    weather_list = data.get("weather", [])
39    temp = main.get("temp")
40    humidity = main.get("humidity")
41    weather_desc = weather_list[0].get("description") if weather_list else ""
42    entry = {
43      "city": name,
44      "temp": int(round(temp)) if temp is not None else None,
45      "humidity": humidity,
46      "weather": weather_desc.title() if isinstance(weather_desc, str) else weather_desc,
47    }
48    return entry
49
50
51  def append_result(entry: dict, filename: str = RESULTS_FILE) -> None:
52    # Load existing array or start a new one
53    results = []
54    if os.path.exists(filename):
55      try:
56        with open(filename, "r", encoding="utf-8") as f:
57          results = json.load(f)
58          if not isinstance(results, list):
59            results = []
60      except Exception:
61        # If file is corrupt or not JSON, overwrite with a fresh list
62        results = []
63    results.append(entry)
```

task18.4.5.py

lab18.4 > task18.4.5.py > ...

```
51  def append_result(entry: dict, filename: str = RESULTS_FILE) -> None:
64    with open(filename, "w", encoding="utf-8") as f:
65      json.dump(results, f, indent=2, ensure_ascii=False)
66
67
68  def print_formatted(entry: dict) -> None:
69    city = entry.get("city", "Unknown")
70    temp = entry.get("temp", "N/A")
71    humidity = entry.get("humidity", "N/A")
72    weather = entry.get("weather", "")
73    print(f"City: {city}")
74    print(f"Temperature: {temp}°C")
75    print(f"Humidity: {humidity}%")
76    print(f"Weather: {weather}")
77
78
79  def main():
80    parser = argparse.ArgumentParser(description="Fetch weather and store results locally.")
81    parser.add_argument("cities", nargs="*", help="City name(s) to fetch (e.g. London).")
82    parser.add_argument("--api-key", "-k", help="OpenWeatherMap API key (or set OPENWEATHER_API_KEY).")
83    parser.add_argument("--file", "-f", default=RESULTS_FILE, help="Results file (default: results.json).")
84    args = parser.parse_args()
85
86    api_key = args.api_key or os.getenv("OPENWEATHER_API_KEY")
87    if not api_key:
88      api_key = input("Enter OpenWeatherMap API key: ").strip()
89    if not api_key:
90      sys.exit("API key is required.")
91
92    cities = args.cities
93    if not cities:
94      city = input("Enter city name: ").strip()
```

```python
def main():
    if not cities:
        city = input("Enter city name: ").strip()
        if not city:
            sys.exit("No city provided.")
        cities = [city]

    for city in cities:
        entry = fetch_weather(city, api_key)
        if entry:
            print_formatted(entry)
            append_result(entry, filename=args.file)


if __name__ == "__main__":
    main()
```

OUTPUT:

```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab18.4/task18.4.5.py
Enter OpenWeatherMap API key:  818d5d28d56a6b76906457db5dc1b86d
Enter city name: hyderabad
City: Hyderabad
Temperature: 20°C
Humidity: 73%
Weather: Mist
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

OBSERVATION:

This program fetches weather information for one or more cities using the OpenWeatherMap API.
It shows each city's temperature, humidity, and weather description in a clean format.
Every time it runs, it also saves the results into a local JSON file, adding each new entry to the list.
The user can enter cities and API keys through command-line arguments or by typing them when prompted.
Overall, it provides an easy way to look up weather data and keep a growing record of all queries.