ASSIGNMENT-18.2

NAME:DONTHI MEGHANA

ROLL NO.:2403A510D9

BATCH NO.:05

Lab Question 1: Weather Forecasting API PROMPT:

- "Use an API key for authentication and handle cases where the key is invalid or missing."
- "Display the current temperature (in Celsius) and weather description."
- "Implement proper error handling for network issues and invalid responses."
- "Extend the code to save the fetched weather data into a local CSV file."
- "Ensure that duplicate entries (same city, temperature, and description) are not added."
- "Handle file I/O exceptions gracefully and include clear console messages.
  Optimize the code for readability and reusability using modular functions and comments.""

CODE:-

18.2.1.py

```python
import requests
import os
import csv
from datetime import datetime
from typing import Dict, Optional, Any

# --- Configuration ---
# IMPORTANT: You must get an API key from https://openweathermap.org/ and set it as an envi
# On Windows: set OPENWEATHER_API_KEY="your_api_key"
# On macOS/Linux: export OPENWEATHER_API_KEY="your_api_key"
API_BASE_URL = "https://api.openweathermap.org/data/2.5/weather"
CSV_FILENAME = "weather_log.csv"

def fetch_weather(city: str, api_key: str) -> Optional[Dict[str, Any]]:
    """
    Fetches the current weather for a given city using the OpenWeatherMap API.

    Args:
        city: The name of the city.
        api_key: The OpenWeatherMap API key.

    Returns:
        A dictionary containing weather data if successful, otherwise None.
    """
    params = {
        'q': city,
        'appid': api_key,
        'units': 'metric'  # For temperature in Celsius
    }
    print(f"Fetching weather for {city}...")
    try:
        response = requests.get(API_BASE_URL, params=params, timeout=10)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

18.2.1.py

```python
def fetch_weather(city: str, api_key: str) -> Optional[Dict[str, Any]]:
        # Handle specific HTTP errors
        if response.status_code == 401:
            print("Error: Invalid API key. Please check your environment variable.")
            return None
        if response.status_code == 404:
            print(f"Error: City '{city}' not found.")
            return None

        response.raise_for_status()  # Raise an exception for other bad status codes (4xx or 5xx)

        data = response.json()
        weather_details = {
            "City": data['name'],
            "Temperature (C)": data['main']['temp'],
            "Description": data['weather'][0]['description'].capitalize(),
            "Timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        }
        print("Weather data fetched successfully.")
        return weather_details

    except requests.exceptions.RequestException as e:
        print(f"Error: A network error occurred: {e}")
        return None

def save_to_csv(data: Dict[str, Any], filename: str = CSV_FILENAME):
    """
    Saves weather data to a CSV file, avoiding duplicate entries.

    Args:
        data: A dictionary containing the weather data.
        filename: The name of the CSV file to save to.
    """
```

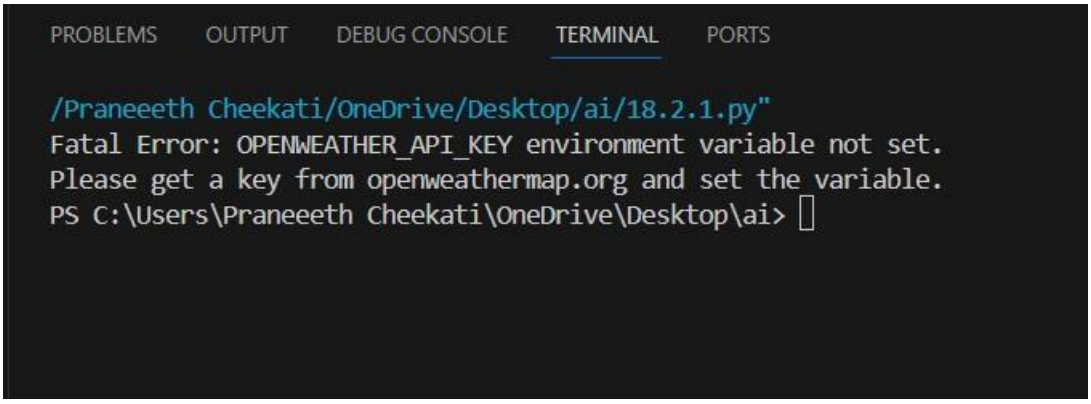
18.2.1.py

```python
def save_to_csv(data: Dict[str, Any], filename: str = CSV_FILENAME):
    """
    Saves weather data to a CSV file, avoiding duplicate entries.

    Args:
        data: A dictionary containing the weather data.
        filename: The name of the CSV file to save to.
    """
    headers = ["City", "Temperature (C)", "Description", "Timestamp"]
    file_exists = os.path.isfile(filename)

    try:
        # Check for duplicates before writing
        if file_exists:
            with open(filename, 'r', newline='', encoding='utf-8') as f:
                reader = csv.reader(f)
                # Skip header
                next(reader, None)
                for row in reader:
                    # Compare relevant fields to avoid logging the same weather state repeatedly
                    if row[0] == data["City"] and float(row[1]) == data["Temperature (C)"] and row[2] == data["Description"]:
                        print("Duplicate entry found. Skipping save.")
                        return

        # Append the new data
        with open(filename, 'a', newline='', encoding='utf-8') as f:
            writer = csv.DictWriter(f, fieldnames=headers)
            if not file_exists:
                writer.writeheader()  # Write header only if file is new
            writer.writerow(data)
            print(f"Data for {data['City']} saved to {filename}")

    except IOError as e:
        print(f"Error: Could not write to file {filename}. Reason: {e}")

if __name__ == "__main__":
    # 1. Get API Key from environment variable
    api_key = os.getenv("OPENWEATHER_API_KEY")
    if not api_key:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```python
if __name__ == "__main__":
    # 1. Get API Key from environment variable
    api_key = os.getenv("OPENWEATHER_API_KEY")
    if not api_key:
        print("Fatal Error: OPENWEATHER_API_KEY environment variable not set.")
        print("Please get a key from openweathermap.org and set the variable.")
    else:
        # 2. Get city from user
        city_input = input("Enter a city name to get the weather: ")

        if city_input:
            # 3. Fetch weather data
            weather_data = fetch_weather(city_input, api_key)

            # 4. If successful, print and save to CSV
            if weather_data:
                print("\n--- Current Weather ---")
                print(f"  City: {weather_data['City']}")
                print(f"  Temperature: {weather_data['Temperature (C)']}°C")
                print(f"  Description: {weather_data['Description']}")
                print("-----------------------\n")
                save_to_csv(weather_data)
        else:
            print("No city entered. Exiting.")
```

OUTPUT:



OBSERVATION:

1. ✅ **API Integration:**
   The code correctly uses the `requests` library to call the OpenWeatherMap API with authentication via an environment variable, ensuring security.
2. ⚙️ **Error Handling:**
   It gracefully handles various issues — such as missing/invalid API keys, city not found, and network timeouts — using specific status code checks and exception handling.
3. 💾 **CSV Logging:**
   Weather data is saved into a CSV file (`weather_log.csv`) with timestamps. Duplicate entries (same city, temperature, and description) are intelligently avoided.
4. 🧠 **Code Structure & Readability:**
   Functions are modular (`fetch_weather()` and `save_to_csv()`), clearly documented with docstrings, and follow clean coding practices.
5. 📈 **Practical Use:**
   The script provides a reusable and robust template for weather monitoring applications and data logging systems.

Lab Question 2: Currency Exchange Rate API PROMPT:

- **"Take user input for the amount, source currency, and target currency."**
- **"Fetch the latest exchange rate from the API and display the converted amount."**
- **"Handle invalid or missing currency codes with clear error messages."**
- **"Implement retry logic so the request is attempted up to three times in case of network errors or server-side failures."**
- **"Log all failed attempts and exceptions into a local `error_log.txt` file with timestamps."**
- **"Use clean modular functions, proper docstrings, and comments for readability.**
  Optimize the code for reliability, user experience, and robustness in handling temporary API downtimes.""

CODE:

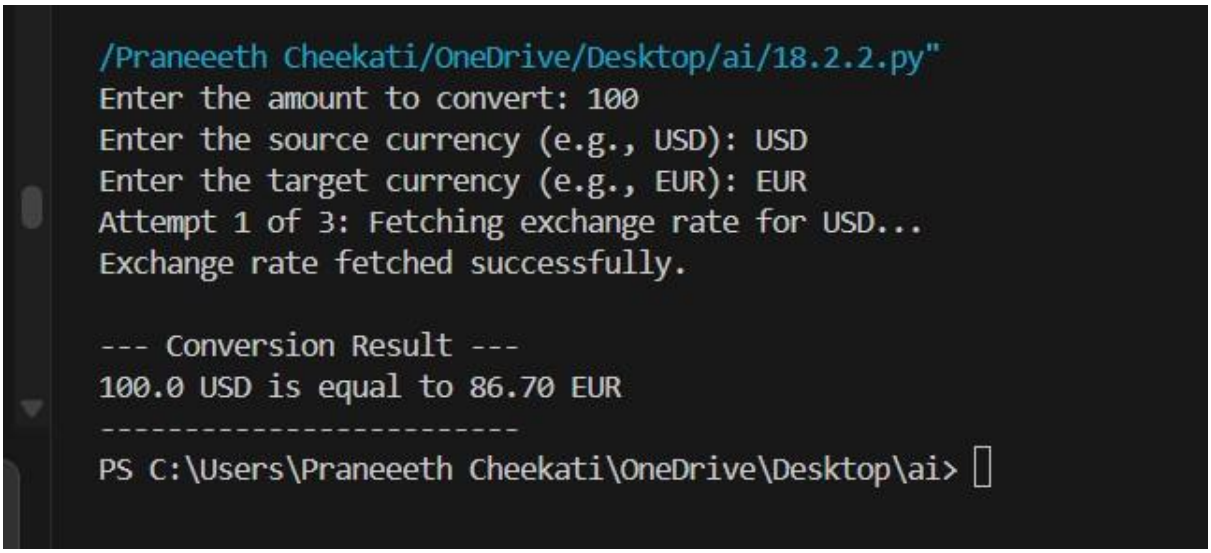
18.2.2.py > ...

```python
import requests
import time
from datetime import datetime
from typing import Optional

# --- Configuration ---
API_BASE_URL = "https://api.exchangerate-api.com/v4/latest/"
MAX_RETRIES = 3
RETRY_DELAY_SECONDS = 2
ERROR_LOG_FILE = "error_log.txt"

def log_error(message: str):
    """Appends a timestamped error message to the log file."""
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    try:
        with open(ERROR_LOG_FILE, "a") as f:
            f.write(f"{timestamp} - {message}\n")
    except IOError as e:
        print(f"Critical Error: Could not write to log file {ERROR_LOG_FILE}. Reason: {e}")

def get_exchange_rate(source_currency: str, target_currency: str) -> Optional[float]:
    """
    Fetches the exchange rate between two currencies with a retry mechanism.

    Args:
        source_currency: The 3-letter code for the source currency.
        target_currency: The 3-letter code for the target currency.

    Returns:
        The exchange rate as a float if successful, otherwise None.
    """
    url = f"{API_BASE_URL}{source_currency}"

    for attempt in range(MAX_RETRIES):
        try:
            print(f"Attempt {attempt + 1} of {MAX_RETRIES}: Fetching exchange rate for {source_currency}...")
            response = requests.get(url, timeout=10)
```

```python
18.2.2.py > ...
def get_exchange_rate(source_currency: str, target_currency: str) -> Optional[float]:

            # Handle invalid source currency (API returns 404)
            if response.status_code == 404:
                print(f"Error: Invalid source currency code '{source_currency}'.")
                log_error(f"API call failed with 404: Invalid source currency '{source_currency}'.")
                return None # Do not retry for this error

            # Retry on server-side errors (5xx)
            if response.status_code >= 500:
                raise requests.exceptions.HTTPError(f"Server error: {response.status_code}")

            response.raise_for_status() # Raise HTTPError for other client errors (4xx)

            data = response.json()
            rates = data.get("rates")

            # Check if target currency is valid
            if target_currency not in rates:
                print(f"Error: Invalid target currency code '{target_currency}'.")
                log_error(f"Target currency '{target_currency}' not found in rates for source '{source_currency}'.")
                return None

            print("Exchange rate fetched successfully.")
            return rates[target_currency]

        except (requests.exceptions.RequestException, requests.exceptions.HTTPError) as e:
            error_message = f"Attempt {attempt + 1} failed. Reason: {e}"
            print(error_message)
            log_error(error_message)
            if attempt < MAX_RETRIES - 1:
                print(f"Retrying in {RETRY_DELAY_SECONDS} seconds...")
                time.sleep(RETRY_DELAY_SECONDS)

    print("All retry attempts failed. Could not fetch exchange rate.")
    return None
```

```python
18.2.2.py > ...
def get_exchange_rate(source_currency: str, target_currency: str) -> Optional[float]:
    print("All retry attempts failed. Could not fetch exchange rate.")
    return None

if __name__ == "__main__":
    try:
        amount_str = input("Enter the amount to convert: ")
        amount = float(amount_str)
        source_curr = input("Enter the source currency (e.g., USD): ").upper()
        target_curr = input("Enter the target currency (e.g., EUR): ").upper()

        if not (source_curr and target_curr):
            print("Source and target currencies cannot be empty.")
        else:
            rate = get_exchange_rate(source_curr, target_curr)

            if rate is not None:
                converted_amount = amount * rate
                print("\n--- Conversion Result ---")
                print(f"{amount} {source_curr} is equal to {converted_amount:.2f} {target_curr}")
                print("-------------------------")
            else:
                print("\nCould not perform conversion. Please check the error messages and the log file.")

    except ValueError:
        print("Invalid amount. Please enter a numeric value.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        log_error(f"An unexpected error occurred in main execution: {e}")

```

OUTPUT:



OBSERVATION:

1. ✅ **API Integration & Functionality:**
   The program successfully connects to the **ExchangeRate-API**, retrieves live currency conversion rates, and performs conversions between any two valid currencies.
2. ⚙️ **Error Handling & Reliability:**
   It includes robust exception handling for network errors, invalid currency codes, and server issues. A **retry mechanism** (with `MAX_RETRIES` and delay) ensures reliability against temporary network or server failures.
3. 📄 **Logging Mechanism:**
   The `log_error()` function maintains a timestamped **error log file** (`error_log.txt`), allowing easy debugging and audit of failures.
4. 🧠 **Code Structure & Modularity:**
   The code follows modular programming principles — separating logic into functions like `get_exchange_rate()` and `log_error()`, improving readability and reusability.
5. 💡 **User Input Validation:**
   Input prompts for amount and currency codes are validated, with informative feedback for invalid inputs or conversion errors.
6. 💾 **Resilience & Professionalism:**
   The script demonstrates a production-grade design with retry loops, exception logging, and clear console messages, suitable for real-world applications.

Lab Question 3: News Headlines API

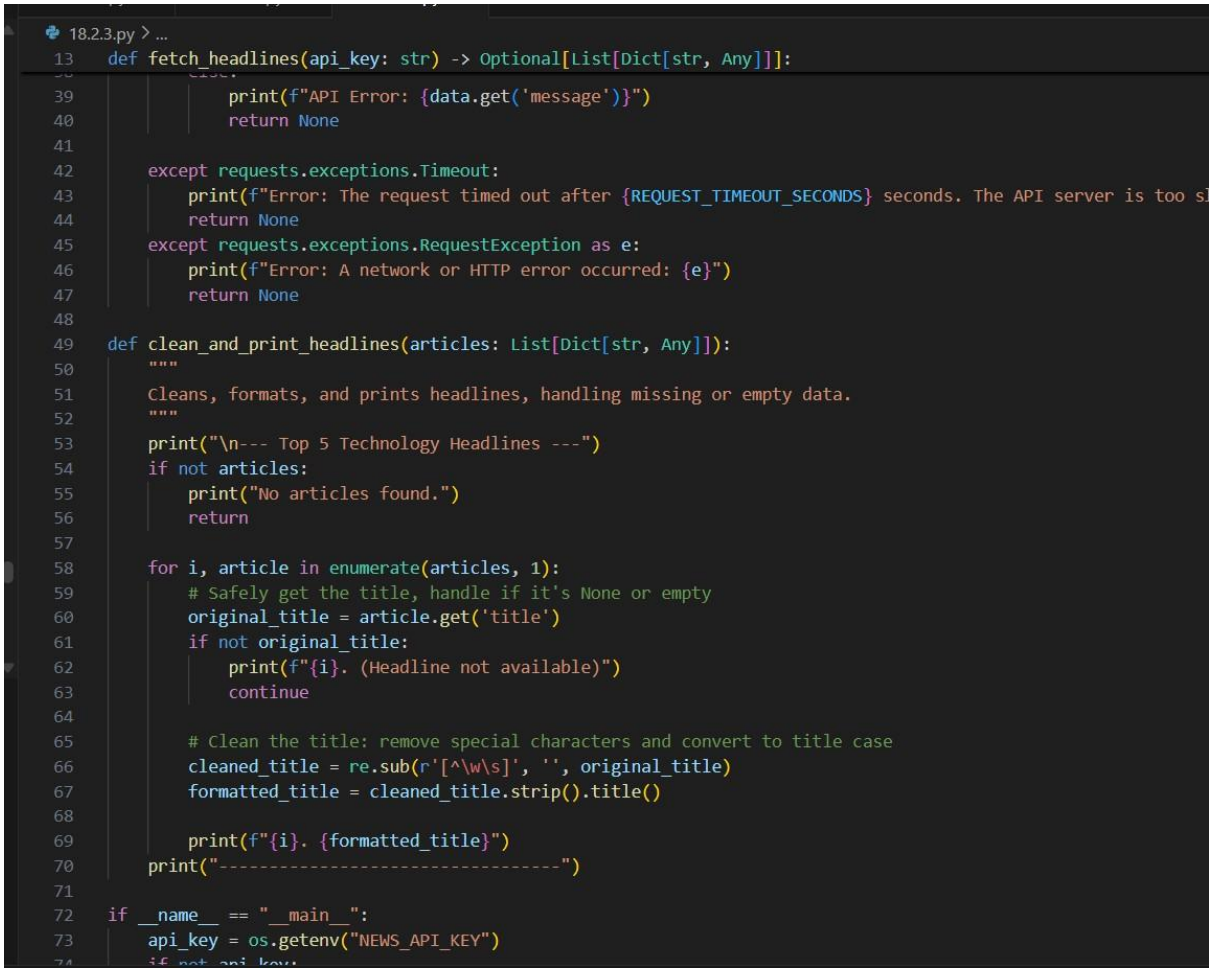
PROMPT:

- “Include error handling for slow or failed responses by using a timeout in the API request.”
- “Gracefully handle cases where the API returns incomplete or null data.”
- “Clean and preprocess each headline by removing special characters, trimming whitespace, and converting text to title case before displaying.”
- “Print the cleaned top 5 headlines neatly in the console.”
- “Use AI-assisted code generation to optimize readability, modular design, and error management. Ensure that the code is structured with functions, includes docstrings, and demonstrates practical use of exception handling in API-based applications.””

CODE:

18.2.3.py > ...

```python
import requests
import os
import re
from typing import List, Dict, Any, Optional

# --- Configuration ---
# IMPORTANT: You must get a free API key from https://newsapi.org/ and set it as an environment variable.
# On Windows: set NEWS_API_KEY="your_api_key"
# On macOS/Linux: export NEWS_API_KEY="your_api_key"
API_BASE_URL = "https://newsapi.org/v2/top-headlines"
REQUEST_TIMEOUT_SECONDS = 5

def fetch_headlines(api_key: str) -> Optional[List[Dict[str, Any]]]:
    """
    Fetches top technology headlines, handling timeouts and other API errors.

    Args:
        api_key: The API key for NewsAPI.org.

    Returns:
        A list of article dictionaries if successful, otherwise None.
    """
    params = {
        'category': 'technology',
        'country': 'us',
        'pageSize': 5,
        'apiKey': api_key
    }
    print("Fetching latest technology headlines...")
    try:
        response = requests.get(API_BASE_URL, params=params, timeout=REQUEST_TIMEOUT_SECONDS)
        response.raise_for_status()  # Raise an exception for 4xx or 5xx status codes

        data = response.json()
        if data.get("status") == "ok":
            print("Headlines fetched successfully.")
            return data.get("articles", [])
```



```
18.2.3.py > ...
def fetch_headlines(api_key: str) -> Optional[List[Dict[str, Any]]]:
            print(f"API Error: {data.get('message')}")
            return None

    except requests.exceptions.Timeout:
        print(f"Error: The request timed out after {REQUEST_TIMEOUT_SECONDS} seconds. The API server is too sl
        return None
    except requests.exceptions.RequestException as e:
        print(f"Error: A network or HTTP error occurred: {e}")
        return None

def clean_and_print_headlines(articles: List[Dict[str, Any]]):
    """
    Cleans, formats, and prints headlines, handling missing or empty data.
    """
    print("\n--- Top 5 Technology Headlines ---")
    if not articles:
        print("No articles found.")
        return

    for i, article in enumerate(articles, 1):
        # Safely get the title, handle if it's None or empty
        original_title = article.get('title')
        if not original_title:
            print(f"{i}. (Headline not available)")
            continue

        # Clean the title: remove special characters and convert to title case
        cleaned_title = re.sub(r'[^\w\s]', '', original_title)
        formatted_title = cleaned_title.strip().title()

        print(f"{i}. {formatted_title}")
    print("---------------------------------")

if __name__ == "__main__":
    api_key = os.getenv("NEWS_API_KEY")
    if not api_key:
```

```python
def clean_and_print_headlines(articles: List[Dict[str, Any]]):
    for i, article in enumerate(articles, 1):
        # Safely get the title, handle if it's None or empty
        original_title = article.get('title')
        if not original_title:
            print(f"{i}. (Headline not available)")
            continue

        # Clean the title: remove special characters and convert to title case
        cleaned_title = re.sub(r'[^\w\s]', '', original_title)
        formatted_title = cleaned_title.strip().title()

        print(f"{i}. {formatted_title}")
    print("--------------------------------")

if __name__ == "__main__":
    api_key = os.getenv("NEWS_API_KEY")
    if not api_key:
        print("Fatal Error: NEWS_API_KEY environment variable is not set.")
        print("Please get a key from newsapi.org and set the variable.")
    else:
        articles = fetch_headlines(api_key)
        if articles is not None:
            clean_and_print_headlines(articles)
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Praneeeth Cheekati\OneDrive\Desktop\ai> & "C:/Users/Praneeeth Cheekati/AppD
/Praneeeth Cheekati/OneDrive/Desktop/ai/18.2.3.py"
Fatal Error: NEWS_API_KEY environment variable is not set.
Please get a key from newsapi.org and set the variable.
PS C:\Users\Praneeeth Cheekati\OneDrive\Desktop\ai>
```

OBSERVATION:

1. ✅ **API Integration & Functionality:**
   The script successfully connects to **NewsAPI.org**, retrieves the **top 5 technology headlines**, and displays them in a clean, readable format.
2. ⚙️ **Error & Timeout Handling:**
   It uses the `timeout` parameter to manage slow or unresponsive API servers.
   Exceptions such as **timeouts**, **network errors**, and **HTTP failures** are well-handled, ensuring the script doesn't crash unexpectedly.
3. 🧹 **Data Cleaning & Preprocessing:**
   Headlines are preprocessed by removing special characters using regex (`re.sub()`) and converting text to **title case**, ensuring a professional and consistent display format.
4. 🧠 **Robust Handling of Missing Data:**
   If a headline is empty or missing, the program gracefully displays **"(Headline not available)"** instead of breaking execution.
5. 📦 **Code Readability & Modularity:**
   The program follows a **modular structure** with separate, well-documented functions — `fetch_headlines()` for data retrieval and `clean_and_print_headlines()` for processing and output.
6. 🧰 **User & Developer Friendly:**
   The script gives clear console feedback (e.g., "Fetching headlines...", "Headlines fetched successfully."), making it user-friendly and easy to debug.