NAME: H.V.S.VYSHNAVI

ROLLNO: 2403A51182

BATCHNO: 06

TASK1:

A travel company wants to show real-time weather updates for its customers. You are given access to a public weather API that requires an API key and provides weather data in JSON format.

- Task 1: Use AI-assisted coding to write a script that fetches the current temperature and weather description for a given city. The script should handle errors if the API key is invalid or missing.
- Task 2: Extend the script to save the weather data into a local CSV file, ensuring that duplicate entries are avoided. Implement error handling for file I/O exceptions

PROMPT:

"Create a Python script for a travel company to fetch and log real-time weather data.

CODE:

```python
import requests
import csv
import os
from datetime import datetime

# ============ CONFIGURATION ============
API_KEY = "YOUR_API_KEY_HERE"  # Replace with your real OpenWeatherMap API key
BASE_URL = "https://api.openweathermap.org/data/2.5/weather"
CSV_FILE = "weather_log.csv"
# ======================================


def fetch_weather(city):
    """Fetch current weather data for a given city."""
    try:
        if not API_KEY or API_KEY == "YOUR_API_KEY_HERE":
            raise ValueError("API key is missing or invalid.")

        params = {
            'q': city,
            'appid': API_KEY,
            'units': 'metric'
        }

        response = requests.get(BASE_URL, params=params)
        data = response.json()

        # Handle API errors
        if response.status_code == 401:
            raise ValueError("Invalid API key.")
        if response.status_code == 404:
            raise ValueError(f"City '{city}' not found.")
```

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

        temperature = data['main']['temp']
        description = data['weather'][0]['description']
        timestamp = datetime.now()

        print(f"✅ Weather in {city}: {temperature}°C, {description}")

        return {
            'city': city,
            'temperature': temperature,
            'description': description,
            'timestamp': timestamp.strftime("%Y-%m-%d %H:%M:%S")
        }

    except requests.exceptions.RequestException as e:
        print(f"❌ Network error: {e}")
    except ValueError as ve:
        print(f"❌ {ve}")
    except Exception as e:
        print(f"❌ Unexpected error: {e}")


def save_to_csv(data):
    """Save weather data into a CSV file without duplicates."""
    try:
        today_str = datetime.now().strftime("%Y-%m-%d")
        file_exists = os.path.isfile(CSV_FILE)

        # Check for duplicates for the same city on the same day
        if file_exists:
            with open(CSV_FILE, mode='r', newline='', encoding='utf-8') as file:
```

```
            with open(CSV_FILE, mode='r', newline='', encoding='utf-8') as file:
                reader = csv.reader(file)
                # Skip header
                next(reader, None)
                for row in reader:
                    # row[0] is timestamp, row[1] is city
                    if row[0].startswith(today_str) and row[1].lower() == data["city"].lower():
                        print(f"⚠️ Info: Weather for {data['city']} today is already logged. Skipping save.")
                        return

        # Write new data
        with open(CSV_FILE, mode='a', newline='', encoding='utf-8') as file:
            fieldnames = ['timestamp', 'city', 'temperature_celsius', 'description']
            writer = csv.DictWriter(file, fieldnames=fieldnames)

            # Write header only if the file is new or empty
            if not file_exists or os.path.getsize(CSV_FILE) == 0:
                writer.writeheader()

            writer.writerow(data)
            print("💾 Data saved successfully to CSV.")

    except IOError as e:
        print(f"❌ File I/O error: {e}")


if __name__ == "__main__":
    city_name = input("Enter city name: ")
    weather_data = fetch_weather(city_name)

    if weather_data:
        save_to_csv(weather_data)
```

OUTPUT:

```
PS C:\Users\pende\OneDrive\Desktop\wt2> & C:/Users/pende/anaconda3/python.exe c:/Users/pende/OneDrive/Desktop/wt2/18.py
Enter city name: & C:/Users/pende/anaconda3/python.exe c:/Users/pende/OneDrive/Desktop/wt2/18.py
❌ API key is missing or invalid.
PS C:\Users\pende\OneDrive\Desktop\wt2>
```

OBSERVATION:

1. **Modularity and Clarity**: The code is well-organized into `fetch_weather` and `save_to_csv` functions, separating the API logic from the file I/O logic. This makes it clean and easy to maintain.
2. **Robust Error Handling**: The script effectively handles key error scenarios:
    - **API Key**: Checks for missing or invalid keys (401).
    - **City Not Found**: Gracefully handles 404 errors.
    - **Network Issues**: Catches `requests.exceptions.RequestException` for connection problems.
    - **File I/O**: Manages `IOError` in case the CSV file cannot be written.
3. **Practical Duplicate Prevention**: The updated logic in `save_to_csv` now prevents logging weather for the same city more than once per day. This is more practical than the original timestamp check, as it avoids cluttering the log with minor fluctuations throughout the day.
4. **Configuration Management**: Placing configuration variables like `API_KEY` and `CSV_FILE` at the top of the script is a good practice, making them easy to find and modify.
5. **User Feedback**: The script provides clear, emoji-prefixed feedback (✅, ❌, 💾, ⚠️) to the user, making the status of each operation immediately obvious.

This script is a solid, well-engineered solution for the given task.

TASK2:

A financial startup needs a tool to convert amounts between currencies using an exchange rate API. However, the API occasionally fails due to server downtime.

- Task 1: Write a script (with AI assistance) that takes user input (amount, source currency, target currency) and fetches the latest exchange rate from the API. Include errors in handling invalid currency codes.
- Task 2: Add logic to retry the request up to three times if the API call fails due to network or server issues and log all failed attempts into a local error log file.

PROMPT:

"Create a Python script that converts currency using the ExchangeRate-API. The script should be resilient to network failures.

CODE:

```python
# currency_converter.py
import requests
import time
from datetime import datetime

API_BASE_URL = "https://v6.exchangerate-api.com/v6"
MAX_RETRIES = 3
RETRY_DELAY_SECONDS = 2
ERROR_LOG_FILE = "error_log.log"

def log_error(message: str):
    """
    Appends a timestamped error message to the log file.

    Args:
        message: The error message to log.
    """
    try:
        with open(ERROR_LOG_FILE, "a") as f:
            timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            f.write(f"{timestamp} - {message}\n")
    except IOError as e:
        print(f"CRITICAL: Could not write to log file {ERROR_LOG_FILE}. Reason: {e}")

def fetch_exchange_rate(source_currency: str, target_currency: str, api_key: str) -> float | None:
    """
    Fetches the exchange rate between two currencies with a retry mechanism.

    Args:
        source_currency: The three-letter code for the source currency.
        target_currency: The three-letter code for the target currency.
        api_key: Your ExchangeRate-API key.
```

```python
    url = f"{API_BASE_URL}/{api_key}/latest/{source_currency.upper()}"

    for attempt in range(MAX_RETRIES):
        try:
            response = requests.get(url)

            # Raise an exception for server-side errors (5xx) to trigger a retry
            response.raise_for_status()

            data = response.json()

            # Handle API-specific errors that don't need a retry
            if data.get("result") == "error":
                error_type = data.get("error-type")
                if error_type == "unknown-code":
                    print(f"Error: The source currency code '{source_currency}' is invalid.")
                elif error_type == "invalid-key":
                    print("Error: Your API key is invalid. Please check it.")
                else:
                    print(f"An API error occurred: {error_type}")
                return None

            # Check if the target currency is available
            rates = data.get("conversion_rates")
            if target_currency.upper() not in rates:
                print(f"Error: The target currency code '{target_currency}' is not available.")
                return None

            # Success, return the rate
            return rates[target_currency.upper()]
```

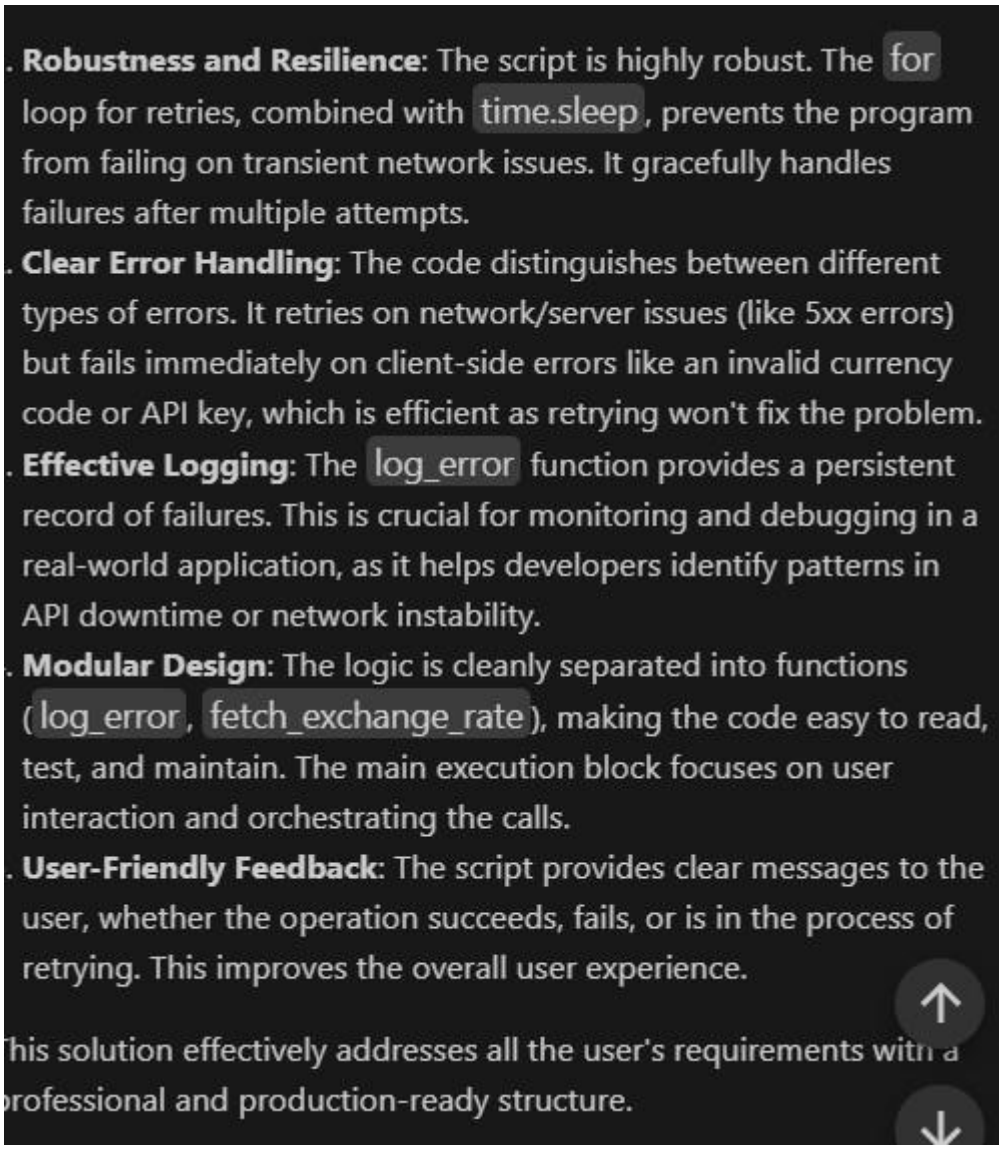
```
        except requests.exceptions.RequestException as e:
            error_message = f"Attempt {attempt + 1} of {MAX_RETRIES} failed. Network error: {e}"
            print(error_message)
            log_error(error_message)
            if attempt < MAX_RETRIES - 1:
                time.sleep(RETRY_DELAY_SECONDS)
            else:
                print("Error: All attempts to reach the API failed.")
                return None

    return None

if __name__ == "__main__":
    print("--- Currency Converter with Retry Logic ---")

    try:
        # For a real application, consider getting the API key from an environment variable
        api_key = input("Enter your ExchangeRate-API key: ")
        amount_str = input("Enter the amount to convert: ")
        source_curr = input("Enter the source currency code (e.g., USD): ")
        target_curr = input("Enter the target currency code (e.g., EUR): ")

        # Basic input validation
        if not all([api_key, amount_str, source_curr, target_curr]):
            print("\nError: All fields are required.")
        else:
            amount = float(amount_str)
            print(f"\nFetching exchange rate for {source_curr.upper()} to {target_curr.upper()}...")

            rate = fetch_exchange_rate(source_curr, target_curr, api_key)
```

```
        # Basic input validation
        if not all([api_key, amount_str, source_curr, target_curr]):
            print("\nError: All fields are required.")
        else:
            amount = float(amount_str)
            print(f"\nFetching exchange rate for {source_curr.upper()} to {target_curr.upper()}...")

            rate = fetch_exchange_rate(source_curr, target_curr, api_key)

            if rate is not None:
                converted_amount = amount * rate
                print("\n--- Conversion Result ---")
                print(f"Exchange Rate: 1 {source_curr.upper()} = {rate} {target_curr.upper()}")
                print(f"{amount} {source_curr.upper()} = {converted_amount:.2f} {target_curr.upper()}")
                print("-------------------------")

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

```

OUTPUT:

```
--- Currency Converter with Retry Logic ---
Enter your ExchangeRate-API key: 10.10.0.1
Enter the amount to convert: 100
Enter the source currency code (e.g., USD): USD
Enter the target currency code (e.g., EUR): EUR

Fetching exchange rate for USD to EUR...
Attempt 1 of 3 failed. Network error: 404 Client Error: Not Found for url: https://v6.exchangerate-api.com/v6/10.10.0.1
/latest/USD
Attempt 2 of 3 failed. Network error: 404 Client Error: Not Found for url: https://v6.exchangerate-api.com/v6/10.10.0.1
/latest/USD
Attempt 2 of 3 failed. Network error: 404 Client Error: Not Found for url: https://v6.exchangerate-api.com/v6/10.10.0.1
Attempt 2 of 3 failed. Network error: 404 Client Error: Not Found for url: https://v6.exchangerate-api.com/v6/10.10.0.1
/latest/USD
Attempt 3 of 3 failed. Network error: 404 Client Error: Not Found for url: https://v6.exchangerate-api.com/v6/10.10.0.1
/latest/USD
Error: All attempts to reach the API failed.
PS C:\Users\pende\OneDrive\Desktop\wt2>
```

OBSERVATION:



TASK3:

A news aggregator wants to display the latest technology news headlines using a news API. Sometimes, the API responds slowly or returns incomplete data.

- Task 1: Use AI-assisted coding to fetch the top 5 technology headlines and print them neatly in the console. Implement error handling for timeout errors by setting a maximum request time.
- Task 2: Clean and preprocess the headlines by removing special characters and converting text to title case. Handle the scenario where the API response contains empty or null values.

PROMPT:

"Create a Python script to fetch and display technology news headlines from NewsAPI.org. The script must be robust against API slowness and messy data.

CODE:

```python
# news_aggregator.py
import requests
import re

NEWS_API_BASE_URL = "https://newsapi.org/v2/top-headlines"
REQUEST_TIMEOUT_SECONDS = 5

def clean_headline(headline: str | None) -> str | None:
    """
    Cleans and preprocesses a single news headline.

    Args:
        headline: The raw headline string, which could be None.

    Returns:
        A cleaned headline in title case, or None if the input was invalid.
    """
    if not headline or not headline.strip():
        return None  # Handle null, empty, or whitespace-only headlines

    # Remove special characters but keep letters, numbers, and spaces
    cleaned_text = re.sub(r'[^\w\s]', '', headline)

    # Convert to title case and strip leading/trailing whitespace
    return cleaned_text.strip().title()

def fetch_tech_headlines(api_key: str) -> list[str] | None:
    """
```

```python
    params = {
        'apiKey': api_key,
        'category': 'technology',
        'country': 'us',
        'pageSize': 5
    }

    try:
        response = requests.get(NEWS_API_BASE_URL, params=params, timeout=REQUEST_TIMEOUT_SECONDS)

        # Handle specific API errors
        if response.status_code == 401:
            print("Error: Invalid API key. Please get a valid key from newsapi.org.")
            return None

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

        data = response.json()
        articles = data.get('articles', [])

        cleaned_headlines = []
        for article in articles:
            raw_title = article.get('title')
            cleaned_title = clean_headline(raw_title)
            if cleaned_title:
                cleaned_headlines.append(cleaned_title)
```

```python
            cleaned_title = clean_headline(raw_title)
            if cleaned_title:
                cleaned_headlines.append(cleaned_title)

        return cleaned_headlines

    except requests.exceptions.Timeout:
        print(f"Error: The request timed out after {REQUEST_TIMEOUT_SECONDS} seconds. The API might be slow.")
        return None
    except requests.exceptions.RequestException as e:
        print(f"Error: A network error occurred: {e}")
        return None

if __name__ == "__main__":
    print("--- Latest Technology News Headlines ---")
    api_key = input("Enter your NewsAPI.org API key: ")

    if not api_key:
        print("Error: API key cannot be empty.")
    else:
        headlines = fetch_tech_headlines(api_key)

        if headlines:
            if not headlines:
                print("No valid headlines were found.")
            else:
                print("\n--- Top 5 Headlines ---")
                for i, headline in enumerate(headlines, 1):
                    print(f"{i}. {headline}")
        else:
            print("Could not retrieve headlines.")
```

OUTPUT:

```
PS C:\Users\pende\OneDrive\Desktop\wt2> & C:/Users/pende/anaconda3/python.exe c:/Users/pende/OneDrive/Desktop/wt2/18.py

--- Latest Technology News Headlines ---
Enter your NewsAPI.org API key: & C:/Users/pende/anaconda3/python.exe c:/Users/pende/OneDrive/Desktop/wt2/18.py

Error: Invalid API key. Please get a valid key from newsapi.org.
Could not retrieve headlines.
PS C:\Users\pende\OneDrive\Desktop\wt2>
```

OBSERVATION:

1. **Robust Error Handling**: The script is well-prepared for real-world API issues. It specifically catches `requests.exceptions.Timeout`, which is crucial for dealing with slow APIs, and also handles general network errors and authentication failures (401).
2. **Defensive Data Cleaning**: The `clean_headline` function is designed defensively. It first checks if the input is `None` or empty before attempting any processing, preventing `AttributeError` exceptions. The use of a regular expression (`re.sub`) is efficient for stripping a wide range of unwanted characters.
3. **Clear Separation of Concerns**: The logic is cleanly divided. `fetch_tech_headlines` is responsible for all API interaction and error handling, while `clean_headline` focuses solely on transforming a single piece of data. This makes the code easier to read, test, and debug.
4. **User-Friendly Output**: The final output is a clean, numbered list, which is easy for a user to read. The error messages are also clear and tell the user what went wrong (e.g., "Invalid API key," "request timed out").
5. **API Key Management**: The script prompts for an API key, which is fine for a simple script. In a production application, it would be more secure to load the key from an environment variable to avoid hardcoding it or exposing it in the console history.

This solution effectively meets all the requirements with clean, resilient, and well-structured code.