

NAME: BAYANA ARJUN

ROLLNO: 2403A510A9

BATCHNO: 05

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:

```
1  import requests
2  import csv
3  import os
4  from datetime import datetime
5
6  # ===== CONFIGURATION =====
7  API_KEY = "YOUR_API_KEY_HERE" # Replace with your real OpenWeatherMap API key
8  BASE_URL = "https://api.openweathermap.org/data/2.5/weather"
9  CSV_FILE = "weather_log.csv"
10 # =====
11
12
13 def fetch_weather(city):
14     """Fetch current weather data for a given city."""
15     try:
16         if not API_KEY or API_KEY == "YOUR_API_KEY_HERE":
17             raise ValueError("API key is missing or invalid.")
18
19         params = {
20             'q': city,
21             'appid': API_KEY,
22             'units': 'metric'
23         }
24
25         response = requests.get(BASE_URL, params=params)
26         data = response.json()
27
28         # Handle API errors
29         if response.status_code == 401:
30             raise ValueError("Invalid API key.")
31         if response.status_code == 404:
32             raise ValueError(f"city '{city}' not found.")
```

```

34     response.raise_for_status() # Raise an exception for other bad responses (4xx or 5xx)
35
36     temperature = data['main']['temp']
37     description = data['weather'][0]['description']
38     timestamp = datetime.now()
39
40     print(f"💡 Weather in {city}: {temperature}°C, {description}")
41
42     return {
43         'city': city,
44         'temperature': temperature,
45         'description': description,
46         'timestamp': timestamp.strftime("%Y-%m-%d %H:%M:%S")
47     }
48
49 except requests.exceptions.RequestException as e:
50     print(f"❌ Network error: {e}")
51 except ValueError as ve:
52     print(f"❌ {ve}")
53 except Exception as e:
54     print(f"❌ Unexpected error: {e}")
55
56
57 def save_to_csv(data):
58     """Save weather data into a CSV file without duplicates."""
59     try:
60         today_str = datetime.now().strftime("%Y-%m-%d")
61         file_exists = os.path.isfile(CSV_FILE)
62
63         # Check for duplicates for the same city on the same day
64         if file_exists:
65             with open(CSV_FILE, mode='r', newline='', encoding='utf-8') as file:
66                 with open(CSV_FILE, mode='r', newline='', encoding='utf-8') as file:
67                     reader = csv.reader(file)
68                     # Skip header
69                     next(reader, None)
70                     for row in reader:
71                         # row[0] is timestamp, row[1] is city
72                         if row[0].startswith(today_str) and row[1].lower() == data["city"].lower():
73                             print("⚠️ Info: Weather for {data['city']} today is already logged. Skipping save.")
74                             return
75
76                     # Write new data
77                     with open(CSV_FILE, mode='a', newline='', encoding='utf-8') as file:
78                         fieldnames = ['timestamp', 'city', 'temperature_celsius', 'description']
79                         writer = csv.DictWriter(file, fieldnames=fieldnames)
80
81                         # Write header only if the file is new or empty
82                         if not file_exists or os.path.getsize(CSV_FILE) == 0:
83                             writer.writeheader()
84
85                         writer.writerow(data)
86                         print("💾 Data saved successfully to CSV.")
87
88     except IOError as e:
89         print(f"❌ File I/O error: {e}")
90
91 if __name__ == "__main__":
92     city_name = input("Enter city name: ")
93     weather_data = fetch_weather(city_name)
94
95     if weather_data:
96         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:

```

1  # currency_converter.py
2  import requests
3  import time
4  from datetime import datetime
5
6  API_BASE_URL = "https://v6.exchangerate-api.com/v6"
7  MAX_RETRIES = 3
8  RETRY_DELAY_SECONDS = 2
9  ERROR_LOG_FILE = "error_log.log"
10
11 def log_error(message: str):
12     """
13         Appends a timestamped error message to the log file.
14
15     Args:
16         message: The error message to log.
17     """
18
19     try:
20         with open(ERROR_LOG_FILE, "a") as f:
21             timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
22             f.write(f"{timestamp} - {message}\n")
23     except IOError as e:
24         print(f"CRITICAL: Could not write to log file {ERROR_LOG_FILE}. Reason: {e}")
25
26 def fetch_exchange_rate(source_currency: str, target_currency: str, api_key: str) -> float | None:
27     """
28         Fetches the exchange rate between two currencies with a retry mechanism.
29
30     Args:
31         source_currency: The three-letter code for the source currency.
32         target_currency: The three-letter code for the target currency.
33         api_key: Your ExchangeRate-API key.
34
35     url = f"{API_BASE_URL}/{api_key}/latest/{source_currency.upper()}"
36
37     for attempt in range(MAX_RETRIES):
38         try:
39             response = requests.get(url)
40
41             # Raise an exception for server-side errors (5xx) to trigger a retry
42             response.raise_for_status()
43
44             data = response.json()
45
46             # Handle API-specific errors that don't need a retry
47             if data.get("result") == "error":
48                 error_type = data.get("error-type")
49                 if error_type == "unknown-code":
50                     print(f"Error: The source currency code '{source_currency}' is invalid.")
51                 elif error_type == "invalid-key":
52                     print("Error: Your API key is invalid. Please check it.")
53                 else:
54                     print(f"An API error occurred: {error_type}")
55             return None
56
57             # Check if the target currency is available
58             rates = data.get("conversion_rates")
59             if target_currency.upper() not in rates:
60                 print(f"Error: The target currency code '{target_currency}' is not available.")
61                 return None
62
63             # Success, return the rate
64             return rates[target_currency.upper()]
65
66
67

```

```

37     url = f"{API_BASE_URL}/{api_key}/latest/{source_currency.upper()}"
38
39     for attempt in range(MAX_RETRIES):
40         try:
41             response = requests.get(url)
42
43             # Raise an exception for server-side errors (5xx) to trigger a retry
44             response.raise_for_status()
45
46             data = response.json()
47
48             # Handle API-specific errors that don't need a retry
49             if data.get("result") == "error":
50                 error_type = data.get("error-type")
51                 if error_type == "unknown-code":
52                     print(f"Error: The source currency code '{source_currency}' is invalid.")
53                 elif error_type == "invalid-key":
54                     print("Error: Your API key is invalid. Please check it.")
55                 else:
56                     print(f"An API error occurred: {error_type}")
57             return None
58
59             # Check if the target currency is available
60             rates = data.get("conversion_rates")
61             if target_currency.upper() not in rates:
62                 print(f"Error: The target currency code '{target_currency}' is not available.")
63                 return None
64
65             # Success, return the rate
66             return rates[target_currency.upper()]
67
68
69

```

```

67
68     except requests.exceptions.RequestException as e:
69         error_message = f"Attempt {attempt + 1} of {MAX_RETRIES} failed. Network error: {e}"
70         print(error_message)
71         log_error(error_message)
72         if attempt < MAX_RETRIES - 1:
73             time.sleep(RETRY_DELAY_SECONDS)
74         else:
75             print("Error: All attempts to reach the API failed.")
76             return None
77
78     return None
79
80 if __name__ == "__main__":
81     print("--- Currency Converter with Retry Logic ---")
82
83     try:
84         # For a real application, consider getting the API key from an environment variable
85         api_key = input("Enter your ExchangeRate-API key: ")
86         amount_str = input("Enter the amount to convert: ")
87         source_curr = input("Enter the source currency code (e.g., USD): ")
88         target_curr = input("Enter the target currency code (e.g., EUR): ")
89
90         # Basic input validation
91         if not all([api_key, amount_str, source_curr, target_curr]):
92             print("\nError: All fields are required.")
93         else:
94             amount = float(amount_str)
95             print(f"\nFetching exchange rate for {source_curr.upper()} to {target_curr.upper()}...")
96
97             rate = fetch_exchange_rate(source_curr, target_curr, api_key)
98
99
100            # Basic input validation
101            if not all([api_key, amount_str, source_curr, target_curr]):
102                print("\nError: All fields are required.")
103            else:
104                amount = float(amount_str)
105                print(f"\nFetching exchange rate for {source_curr.upper()} to {target_curr.upper()}...")
106
107                rate = fetch_exchange_rate(source_curr, target_curr, api_key)
108
109                if rate is not None:
110                    converted_amount = amount * rate
111                    print("\n--- Conversion Result ---")
112                    print(f"Exchange Rate: 1 {source_curr.upper()} = {rate} {target_curr.upper()}")
113                    print(f"{amount} {source_curr.upper()} = {converted_amount:.2f} {target_curr.upper()}")
114                    print("-----")
115
116            except ValueError:
117                print("\nError: Invalid amount. Please enter a numeric value.")
118            except Exception as e:
119                print(f"\nAn unexpected error occurred: {e}")
120                log_error(f"An unexpected error occurred: {e}")
121
122

```

OUTPUT:

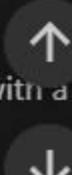
```
--- Currency Converter with Retry Logic ---
Enter your ExchangeRate-API key: 10.10.0.1
Enter the amount to convert: 100
Enter the source currency code (e.g., USD): USD
Enter the target currency code (e.g., EUR): EUR

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/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:

- . **Robustness and Resilience:** The script is highly robust. The `for` loop for retries, combined with `time.sleep`, prevents the program from failing on transient network issues. It gracefully handles failures after multiple attempts.
- . **Clear Error Handling:** The code distinguishes between different types of errors. It retries on network/server issues (like 5xx errors) but fails immediately on client-side errors like an invalid currency code or API key, which is efficient as retrying won't fix the problem.
- . **Effective Logging:** The `log_error` function provides a persistent record of failures. This is crucial for monitoring and debugging in a real-world application, as it helps developers identify patterns in API downtime or network instability.
- . **Modular Design:** The logic is cleanly separated into functions (`log_error`, `fetch_exchange_rate`), making the code easy to read, test, and maintain. The main execution block focuses on user interaction and orchestrating the calls.
- . **User-Friendly Feedback:** The script provides clear messages to the user, whether the operation succeeds, fails, or is in the process of retrying. This improves the overall user experience.

This solution effectively addresses all the user's requirements with a professional and production-ready structure.



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:

```
1 # news_aggregator.py
2 import requests
3 import re
4
5 NEWS_API_BASE_URL = "https://newsapi.org/v2/top-headlines"
6 REQUEST_TIMEOUT_SECONDS = 5
7
8 def clean_headline(headline: str | None) -> str | None:
9     """
10     Cleans and preprocesses a single news headline.
11
12     Args:
13         headline: The raw headline string, which could be None.
14
15     Returns:
16         A cleaned headline in title case, or None if the input was invalid.
17     """
18     if not headline or not headline.strip():
19         return None # Handle null, empty, or whitespace-only headlines
20
21     # Remove special characters but keep letters, numbers, and spaces
22     cleaned_text = re.sub(r'[^\\w\\s]', '', headline)
23
24     # Convert to title case and strip leading/trailing whitespace
25     return cleaned_text.strip().title()
26
27 def fetch_tech_headlines(api_key: str) -> list[str] | None:
28     """
```

```

37     params = {
38         'apiKey': api_key,
39         'category': 'technology',
40         'country': 'us',
41         'pageSize': 5
42     }
43
44     try:
45         response = requests.get(NEWS_API_BASE_URL, params=params, timeout=REQUEST_TIMEOUT_SECONDS)
46
47         # Handle specific API errors
48         if response.status_code == 401:
49             print("Error: Invalid API key. Please get a valid key from newsapi.org.")
50             return None
51
52         response.raise_for_status() # Raise an exception for other bad responses (4xx or 5xx)
53
54         data = response.json()
55         articles = data.get('articles', [])
56
57         cleaned_headlines = []
58         for article in articles:
59             raw_title = article.get('title')
60             cleaned_title = clean_headline(raw_title)
61             if cleaned_title:
62                 cleaned_headlines.append(cleaned_title)
63
64             cleaned_title = clean_headline(raw_title)
65             if cleaned_title:
66                 cleaned_headlines.append(cleaned_title)
67
68         return cleaned_headlines
69
70     except requests.exceptions.Timeout:
71         print(f"Error: The request timed out after {REQUEST_TIMEOUT_SECONDS} seconds. The API might be slow.")
72         return None
73     except requests.exceptions.RequestException as e:
74         print(f"Error: A network error occurred: {e}")
75         return None
76
77     if __name__ == "__main__":
78         print("--- Latest Technology News Headlines ---")
79         api_key = input("Enter your NewsAPI.org API key: ")
80
81         if not api_key:
82             print("Error: API key cannot be empty.")
83         else:
84             headlines = fetch_tech_headlines(api_key)
85
86             if headlines:
87                 if not headlines:
88                     print("No valid headlines were found.")
89                 else:
90                     print("\n--- Top 5 Headlines ---")
91                     for i, headline in enumerate(headlines, 1):
92                         print(f"{i}. {headline}")
93             else:
94                 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.