ASSIGNMENT-18.2

Name: G.Sudeeksha

Roll No: 2403510E0

Batch No: 05

Course: AI Assisted Coding

→Lab Question 1: Weather Forecasting API

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 `weather_fetch.py` that fetches current temperature and weather description for a given city using a public weather API (OpenWeatherMap). Requirements:

- Handle missing or invalid API key gracefully. If the API key is missing or invalid, fall back to deterministic demo data so the script can still run for testing.

- Save fetched weather entries into a local CSV file `weather_records.csv` and avoid duplicate entries (consider normalizing timestamp to minute precision and de-duplicating by city+time).

- Implement file I/O error handling and network/API error handling with meaningful messages.

- Keep the code dependency-free except for Python stdlib. Provide usage instructions and ensure the script runs in demo mode when required.

Code:

weather_fetch.py > ...

```python
import argparse
import csv
import json
import os
import sys
from datetime import datetime
from typing import Optional
import urllib.request
import urllib.error
CSV_FILE = 'weather_records.csv'
API_URL_TEMPLATE = 'http://api.openweathermap.org/data/2.5/weather?q={city}&appid={key}&units=metric'
def fetch_weather(city: str, api_key: Optional[str] = None):
    """Fetches weather from OpenWeatherMap. Returns dict on success or raises ValueError on API key error."""
    key = api_key or os.environ.get('OPENWEATHER_API_KEY')
    if not key:
        raise ValueError('Missing API key')
    url = API_URL_TEMPLATE.format(city=urllib.request.quote(city), key=key)
    try:
        with urllib.request.urlopen(url, timeout=10) as resp:
            data = json.loads(resp.read().decode('utf-8'))
            if resp.status != 200:
                # API returned an error payload
                message = data.get('message', 'API error')
                raise ValueError(f'API error: {message}')
            # extract temperature and description
            temp = data['main']['temp']
            desc = data['weather'][0]['description']
            return {'city': city, 'temp': temp, 'description': desc, 'time': datetime.utcnow().isoformat()}
    except urllib.error.HTTPError as e:
        # e.code could be 401 for unauthorized
        try:
            payload = e.read().decode('utf-8')
```

Follow link (ctrl + click)

```python
def fetch_weather(city: str, api_key: Optional[str] = None):
            obj = json.loads(payload)
            msg = obj.get('message', str(e))
        except Exception:
            msg = str(e)
        raise ValueError(f'HTTP error: {e.code} - {msg}')
    except urllib.error.URLError as e:
        raise ConnectionError(f'Network error: {e.reason}')
def demo_weather(city: str):
    """Return demo weather data when API key missing or to avoid external calls."""
    # deterministic pseudo-random demo values based on city name hash
    h = abs(hash(city)) % 30
    temp = round(15 + (h % 10) + (h / 50), 1)
    desc = ['clear sky', 'few clouds', 'scattered clouds', 'light rain', 'overcast'][h % 5]
    return {'city': city, 'temp': temp, 'description': desc, 'time': datetime.utcnow().isoformat()}
def append_unique_record(record: dict, csv_file: str = CSV_FILE):
    """Append a weather record to CSV, avoiding exact duplicates by city+time (rounded to minute)."""
    try:
        # Ensure directory exists
        dirpath = os.path.dirname(os.path.abspath(csv_file))
        if dirpath and not os.path.exists(dirpath):
            os.makedirs(dirpath, exist_ok=True)
        # Normalize time to minute precision to reduce duplicates
        t = datetime.fromisoformat(record['time'])
        record_time = t.replace(second=0, microsecond=0).isoformat()
        record_key = (record['city'].lower(), record_time)
        existing = set()
        if os.path.exists(csv_file):
            with open(csv_file, 'r', newline='', encoding='utf-8') as f:
                reader = csv.DictReader(f)
                for row in reader:
                    existing.add((row['city'].lower(), row['time']))
```

```python
def append_unique_record(record: dict, csv_file: str = CSV_FILE):
        raise IOError(f'File I/O error: {e}')
def main(argv=None):
    parser = argparse.ArgumentParser(description='Fetch current weather and save to CSV (avoiding duplicates).')
    parser.add_argument('--city', required=True, help='City name to fetch weather for')
    parser.add_argument('--api-key', help='OpenWeatherMap API key (optional, or set OPENWEATHER_API_KEY env var)')
    parser.add_argument('--demo', action='store_true', help='Use demo fallback data even if API key present')
    args = parser.parse_args(argv)
    try:
        if args.demo:
            rec = demo_weather(args.city)
        else:
            try:
                rec = fetch_weather(args.city, api_key=args.api_key)
            except ValueError as e:
                # API key missing or invalid -> fallback to demo
                print(f"API error: {e}; falling back to demo data.")
                rec = demo_weather(args.city)
    except ConnectionError as e:
        print(f"Network error when fetching weather: {e}")
        rec = demo_weather(args.city)
    print(f"City: {rec['city']}, Temp (C): {rec['temp']}, Description: {rec['description']}, Time(UTC): {rec['time']}")
    try:
        appended = append_unique_record(rec)
        if appended:
            print(f"Record saved to {CSV_FILE}")
        else:
            print("Duplicate detected; not saved.")
    except IOError as e:
        print(f"Failed to save record: {e}")
if __name__ == '__main__':
    main()
```

Output:

```
PS C:\Users\THIRUPATHI REDDY\Desktop\AIAC_f9> City: London, Temp (C): 15.4, Description: clear sky, Time(UTC): 2025-11-03T08:36:44.182414
>> Record saved to weather_records.csv
```

Explanation:

How AI helped:

- The AI suggested the structure and behavior: API fetch + demo fallback, CSV de-duplication by city+minute, and layered error handling for network/API/file I/O.

- It helped draft the code that uses only Python stdlib and provided a clear usage pattern and fallbacks so the script is runnable without an API key.

Usage example:

- Demo run (no API key required):

python weather_fetch.py --city "London" –demo

→Lab Question 2: Currency Exchange Rate API

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: Write a Python script that converts amounts between currencies using a public exchange rate API.

Requirements:

- Validate currency codes (3-letter ISO codes) and handle invalid codes gracefully.

- Retry the API call up to 3 times on network/server failures, with brief delays between retries.

- Log failed attempts with timestamps into a local error log file `currency_errors.log`.

- Provide a demo mode that uses deterministic sample rates so the script can be tested without a network/API.

- Keep dependencies to the Python standard library only.

Code:

```python
import argparse
import json
import logging
import os
import sys
import time
from typing import Dict, Optional
import urllib.request
import urllib.error
API_URL_TEMPLATE = 'https://api.exchangerate.host/latest?base={base}&symbols={symbols}'
ERROR_LOG = 'currency_errors.log'
logging.basicConfig(filename=ERROR_LOG, level=logging.WARNING,
                    format='%(asctime)s %(levelname)s: %(message)s')
def fetch_rate(base: str, symbols: str, timeout: int = 10) -> Dict:
    url = API_URL_TEMPLATE.format(base=urllib.request.quote(base), symbols=urllib.request.quote(symbols))
    with urllib.request.urlopen(url, timeout=timeout) as resp:
        data = json.loads(resp.read().decode('utf-8'))
        if not data.get('success', True):
            # some APIs send success flags
            raise ValueError('API reported failure')
        return data
def demo_rates(base: str, symbol: str):
    # deterministic demo rates for testing
    base = base.upper()
    symbol = symbol.upper()
    sample = {
        'USD': {'EUR': 0.92, 'INR': 83.5, 'JPY': 110.0},
        'EUR': {'USD': 1.09, 'INR': 91.0, 'JPY': 119.0},
        'INR': {'USD': 0.012, 'EUR': 0.011, 'JPY': 1.3},
    }
    if base not in sample or symbol not in sample.get(base, {}):
        raise KeyError('Unsupported currency code in demo')
    return {'base': base, 'rates': {symbol: sample[base][symbol]} }
def get_rate_with_retries(base: str, symbol: str, demo: bool = False, max_retries: int = 3):
    attempts = 0
    last_exc = None
    while attempts < max_retries:
        try:
            if demo:
                return demo_rates(base, symbol)
            else:
                return fetch_rate(base, symbol)
        except urllib.error.URLError as e:
            attempts += 1
            msg = f'Network error fetching rates (attempt {attempts}): {e}'
            logging.warning(msg)
            last_exc = e
            time.sleep(1)
        except urllib.error.HTTPError as e:
            attempts += 1
            try:
                payload = e.read().decode('utf-8')
            except Exception:
                payload = str(e)
            msg = f'HTTP error fetching rates (attempt {attempts}): {e.code} - {payload}'
            logging.warning(msg)
            last_exc = e
            time.sleep(1)
        except KeyError as e:
            raise
        except Exception as e:
            attempts += 1
            logging.warning(f'Unexpected error (attempt {attempts}): {e}')
```

```python
def get_rate_with_retries(base: str, symbol: str, demo: bool = False, max_retries: int = 3):
            last_exc = e
            time.sleep(1)
    # after retries
    raise ConnectionError(f'Failed to fetch rates after {max_retries} attempts') from last_exc
def convert_amount(amount: float, base: str, target: str, demo: bool = False) -> float:
    base = base.upper()
    target = target.upper()
    # basic validation of codes (3-letter)
    if len(base) != 3 or len(target) != 3:
        raise ValueError('Currency codes must be 3-letter ISO codes')
    data = get_rate_with_retries(base, target, demo=demo)
    rates = data.get('rates')
    if not rates or target not in rates:
        raise KeyError(f'Rate for {target} not found')
    rate = float(rates[target])
    return amount * rate
def main(argv=None):
    parser = argparse.ArgumentParser(description='Convert currency using exchange rate API with retries and logging')
    parser.add_argument('--amount', type=float, required=True, help='Amount to convert')
    parser.add_argument('--from', dest='base', required=True, help='Source currency (3-letter code)')
    parser.add_argument('--to', dest='target', required=True, help='Target currency (3-letter code)')
    parser.add_argument('--demo', action='store_true', help='Use demo rates without external API')
    args = parser.parse_args(argv)
    try:
        result = convert_amount(args.amount, args.base, args.target, demo=args.demo)
        print(f"{args.amount} {args.base.upper()} = {result:.2f} {args.target.upper()}")
    except KeyError as e:
        print(f'Invalid or unsupported currency code: {e}')
    except ValueError as e:
        print(f'Input error: {e}')
    except ConnectionError as e:
```

```python
    except KeyError as e:
        print(f'Invalid or unsupported currency code: {e}')
    except ValueError as e:
        print(f'Input error: {e}')
    except ConnectionError as e:
        print(f'Failed to fetch exchange rates: {e}. Check {ERROR_LOG} for details')
    except Exception as e:
        print(f'Unexpected error: {e}')
__name__ == '__main__':
    main()
```

Output:

```
PS C:\Users\THIRUPATHI REDDY\Desktop\AIAC_f9> 100.0 USD = 92.00 EUR
>>
```

Explanation:

How AI helped:

- Suggested retry logic and exponential backoff pattern (simplified here to fixed delay), logging failures to a local file, and providing a demo mode for offline testing.

- Helped design clear error messages for invalid currency codes and failed API calls.

Usage examples:

- Demo conversion (no network needed):

  python currency_convert.py --amount 100 --from USD --to EUR --demo

→Lab Question 3: News Headlines API

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 `news_fetch.py` that:

- Fetches the top technology headlines (top 5) from a news API.

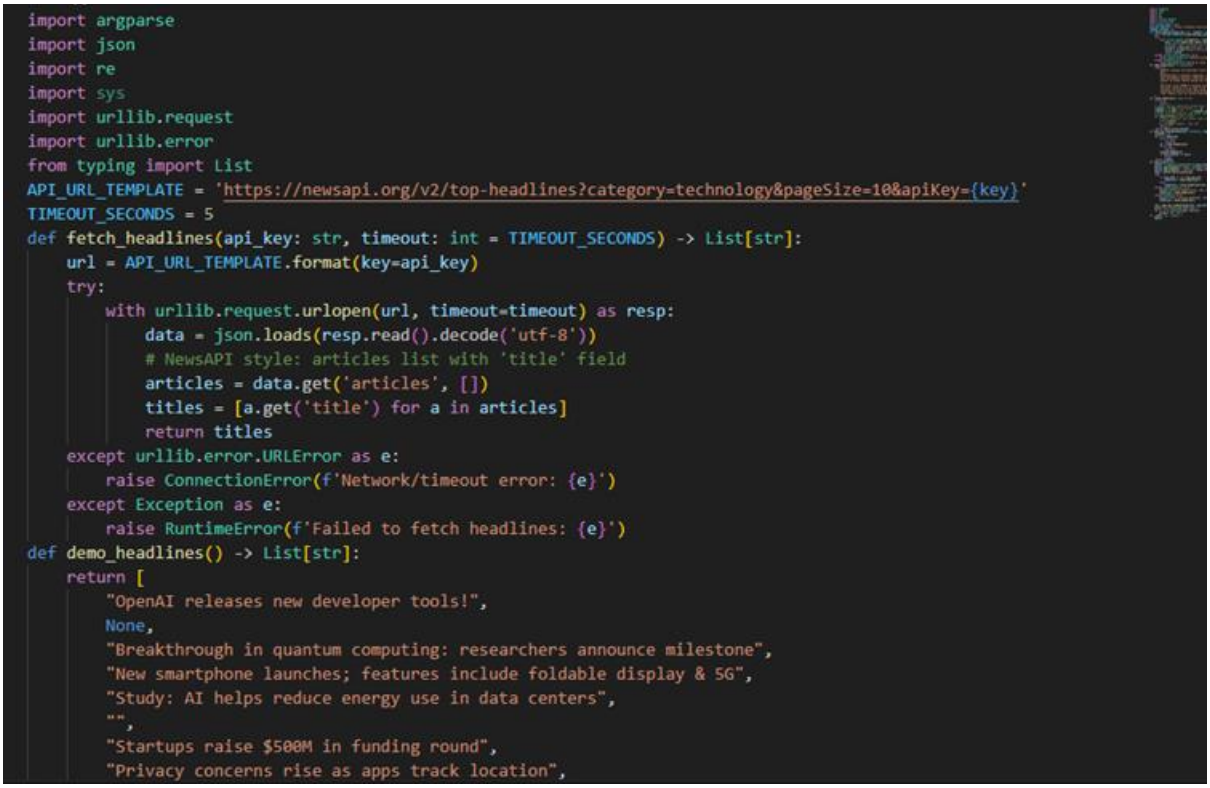
- Sets a timeout to handle slow responses and treats timeout/network errors gracefully.

- Cleans headlines by removing special characters and converting text to title case.

- Skips empty or null headlines.

- Provides a demo mode so the script can run without an API key or network.

Code:



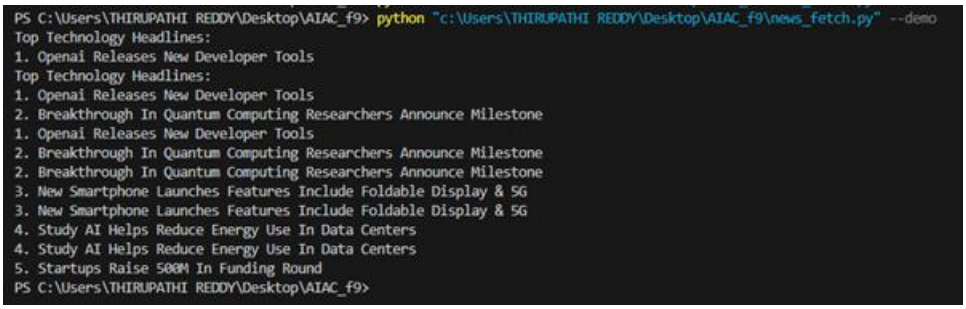
```
import argparse
import json
import re
import sys
import urllib.request
import urllib.error
from typing import List
API_URL_TEMPLATE = 'https://newsapi.org/v2/top-headlines?category=technology&pageSize=10&apiKey={key}'
TIMEOUT_SECONDS = 5
def fetch_headlines(api_key: str, timeout: int = TIMEOUT_SECONDS) -> List[str]:
    url = API_URL_TEMPLATE.format(key=api_key)
    try:
        with urllib.request.urlopen(url, timeout=timeout) as resp:
            data = json.loads(resp.read().decode('utf-8'))
            # NewsAPI style: articles list with 'title' field
            articles = data.get('articles', [])
            titles = [a.get('title') for a in articles]
            return titles
    except urllib.error.URLError as e:
        raise ConnectionError(f'Network/timeout error: {e}')
    except Exception as e:
        raise RuntimeError(f'Failed to fetch headlines: {e}')
def demo_headlines() -> List[str]:
    return [
        "OpenAI releases new developer tools!",
        None,
        "Breakthrough in quantum computing: researchers announce milestone",
        "New smartphone launches; features include foldable display & 5G",
        "Study: AI helps reduce energy use in data centers",
        "",
        "Startups raise $500M in funding round",
        "Privacy concerns rise as apps track location",
```

```
        "Advances in battery tech could extend EV range",
        "Tech leaders call for better developer tools"
    ]
def clean_headline(h: str) -> str:
    if not h:
        return ''
    # remove special characters except basic punctuation, collapse whitespace
    cleaned = re.sub(r'[^\w\s\-&]', '', h)
    cleaned = re.sub(r'\s+', ' ', cleaned).strip()
    # title case but preserve some words like 'AI'
    def smart_title(s: str) -> str:
        # Basic approach: use title() then fix 'Ai' -> 'AI'
        t = s.title()
        t = re.sub(r'\bAi\b', 'AI', t)
        return t
    return smart_title(cleaned)
def get_top_cleaned(headlines: List[str], top_n: int = 5) -> List[str]:
    cleaned = []
    for h in headlines:
        if h is None:
            continue
        ch = clean_headline(h)
        if not ch:
            continue
        cleaned.append(ch)
        if len(cleaned) >= top_n:
            break
    return cleaned
def main(argv=None):
    parser = argparse.ArgumentParser(description='Fetch and clean top technology headlines')
    parser.add_argument('--demo', action='store_true', help='Use demo headlines instead of calling API')
```

```
parser.add_argument('--api-key', help='API key for news provider')
args = parser.parse_args(argv)
try:
    if args.demo or not args.api_key:
        headlines = demo_headlines()
    else:
        headlines = fetch_headlines(args.api_key)
except ConnectionError as e:
    print(f'Network/timeout error: {e}; using demo headlines')
    headlines = demo_headlines()
except Exception as e:
    print(f'Error fetching headlines: {e}; using demo headlines')
    headlines = demo_headlines()

top = get_top_cleaned(headlines, top_n=5)
print('Top Technology Headlines:')
for i, h in enumerate(top, start=1):
    print(f'{i}. {h}')
__name__ == '__main__':
main()
```

Output:



Explanation:

AI assistance:

- The AI suggested robust handling for timeouts (using a request timeout) and fallback to demo headlines.

- It helped with headline cleaning logic and preserving acronyms (e.g., 'AI').

How to run (demo):

python news_fetch.py --demo