SR UNIVERSITY

Program	: B.tech(CSE)
Specialization	:AIML
Course Title	:AI Assisted Coding
Course Code	:24CS002PC215
Semester	:3rd semester
Academic Session	:2025-2026
Name of Student	: A.MANOJ
Enrollment No.	:2403A52031
Batch No	:02
Date	:22/10/2025

LAB ASSIGNMENT-18.2

TASK DESCRIPTION-1:

Movie Database API

Task: Connect to a Movie Database API (e.g., OMDb or TMDB) to fetch details of a movie.

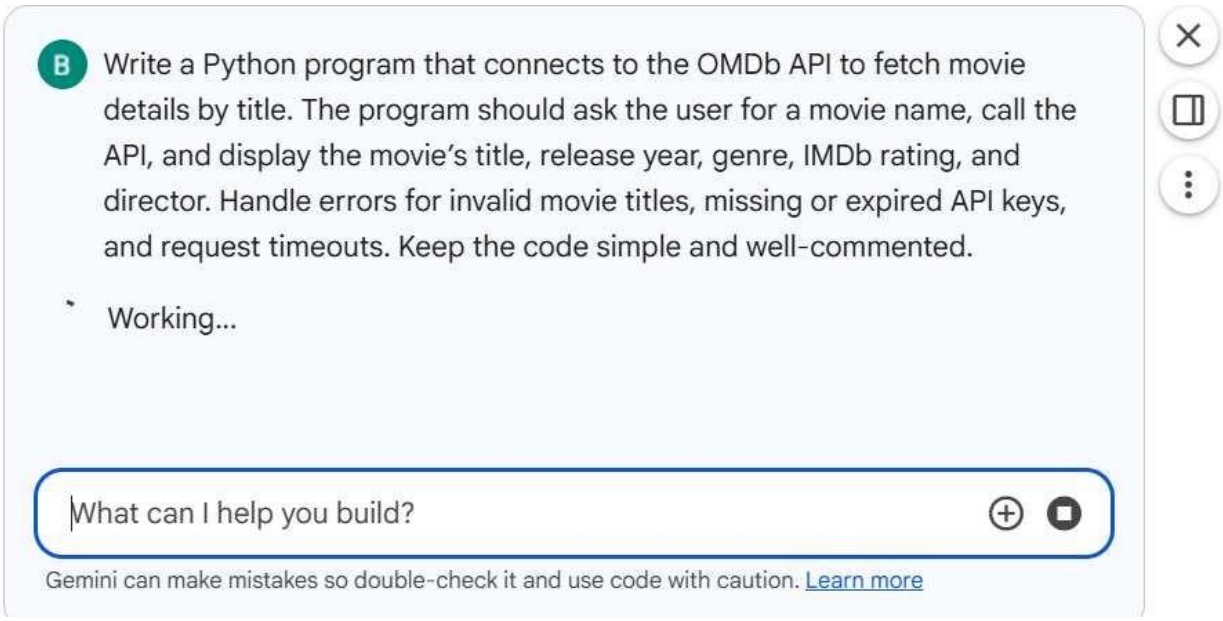
Instructions:

Prompt AI to generate Python code to query the API by movie title. Handle errors like invalid movie name, missing/expired API key, and timeout.

Display title, release year, genre, IMDb rating, and director.

#PROMPT:

Write a Python program that connects to the OMDb API to fetch movie details by title. The program should ask the user for a movie name, call the API, and display the movie's title, release year, genre, IMDb rating, and director. Handle errors for invalid movie titles, missing or expired API keys, and request timeouts. Keep the code simple and well- commented.



#CODE:

```
OMDB_API_KEY = "a7166376"
```

OMDb API Key Setup

To use the OMDb API, you'll need an API key.

1. Go to OMDb API and request a free API key.
2. Once you have your API key, add it to the Colab secrets manager. Click the "🔑" icon in the left panel, and add a new secret with the name `OMDB_API_KEY` and your API key as the value. This ensures your key is stored securely and not directly in your notebook code.

```
# Import necessary libraries
import requests
from google.colab import userdata

# Securely retrieve the OMDb API key from Colab secrets
OMDB_API_KEY = userdata.get('OMDB_API_KEY')
```

```
# Import necessary libraries
import requests
from google.colab import userdata

# Securely retrieve the OMDb API key from Colab secrets
OMDB_API_KEY = userdata.get('OMDB_API_KEY')

# Check if the API key is available
if not OMDB_API_KEY:
    raise ValueError("OMDB_API_KEY not found in Colab secrets. Please add it as instructed above.")

print("OMDb API Key loaded successfully.")
```

```
OMDb API Key loaded successfully.
```

```
def get_movie_details(movie_title: str, api_key: str):
    """
    Fetches movie details from the OMDb API by title.

    Args:
        movie_title (str): The title of the movie to search for.
        api_key (str): Your OMDb API key.

    Returns:
        dict: A dictionary containing movie details if su
```

Toggle Gemini

```
params = {
    't': movie_title, # Search by title
    'apikey': api_key
}

try:
    # Make the API request with a timeout to prevent hanging
    response = requests.get(base_url, params=params, timeout=10)
    response.raise_for_status() # Raise an HTTPError for bad responses (4xx or 5xx)

    data = response.json()

    # Handle OMDb API specific errors
    if data.get('Response') == 'False':
        error_message = data.get('Error', 'Unknown OMDb API error.')
        # Check for API key issues
        if "Invalid API key!" in error_message or "Key expired!" in error_message:
            print(f"Error: {error_message} Please check your OMDb API key.")
        else:
            print(f"Error: {error_message} for movie '{movie_title}'.")
        return None

    # Extract required details
    movie_details = {
        'Title': data.get('Title'),
```

Toggle Gemini

```
            'Title': data.get('Title'),
            'Year': data.get('Year'),
            'Genre': data.get('Genre'),
            'IMDb Rating': data.get('imdbRating'),
            'Director': data.get('Director')
        }
        return movie_details

    except requests.exceptions.Timeout:
        print("Error: Request timed out. Please check your internet connection or try again later.")
        return None
    except requests.exceptions.ConnectionError:
        print("Error: Could not connect to the OMDb API. Check your internet connection.")
        return None
    except requests.exceptions.HTTPError as err:
        print(f"Error: HTTP error occurred: {err}")
        return None
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        return None

# --- Main program execution ---

if __name__ == "__main__":
    # Ask the user for a movie name
```

Toggle Gemini

```python
# --- Main program execution ---

if __name__ == "__main__":
    # Ask the user for a movie name
    user_movie_title = input("Enter the movie title: ")

    if not user_movie_title.strip():
        print("Movie title cannot be empty.")
    else:
        print(f"Fetching details for '{user_movie_title}'...")
        movie_info = get_movie_details(user_movie_title, OMDB_API_KEY)

        if movie_info:
            print("\n--- Movie Details ---")
            for key, value in movie_info.items():
                print(f"{key}: {value}")
        else:
            print("\nCould not retrieve movie details. Please try another title or check the error messages above.")
```

OUTPUT:

```
Enter the movie title: Guardians of the Galaxy Vol. 2
Fetching details for 'Guardians of the Galaxy Vol. 2'...

--- Movie Details ---
Title: Guardians of the Galaxy Vol. 2
Year: 2017
Genre: Action, Adventure, Comedy
IMDb Rating: 7.6
Director: James Gunn
```

TASK DESCRIPTION-2:

Public Transport API

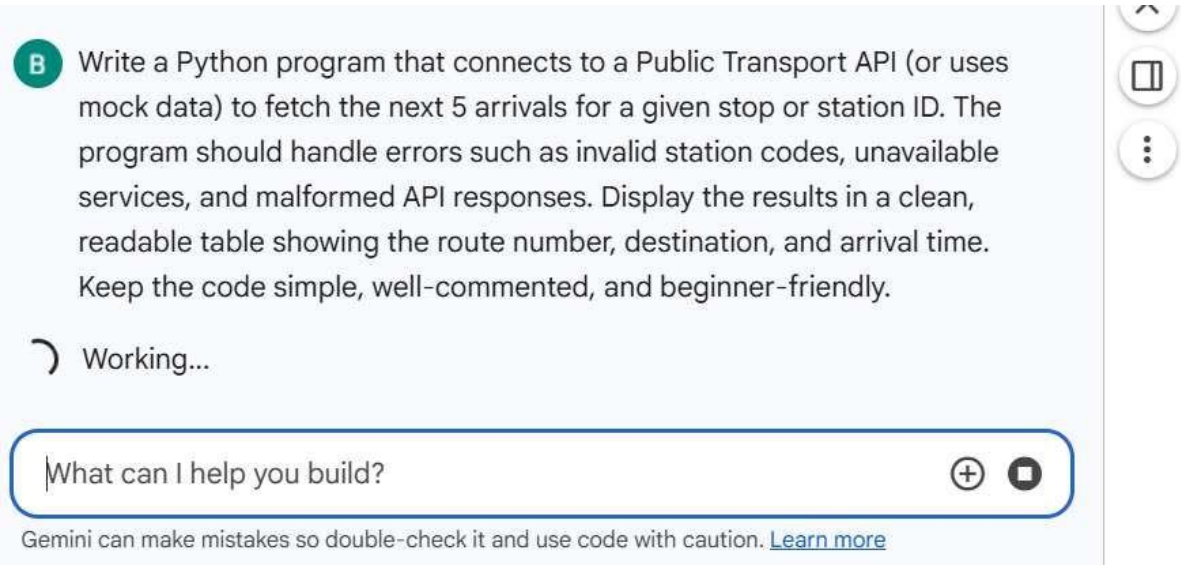
Task: Use a Public Transport API (e.g., city bus/train API or mock data) to fetch live arrival times.

Instructions:

- Fetch the next 5 arrivals for a given stop/station ID.
- Handle invalid station codes, unavailable service, and malformed responses.
- Display results in a readable table with route number, destination, and arrival time

#PROMPT:

Write a Python program that connects to a Public Transport API (or uses mock data) to fetch the next 5 arrivals for a given stop or station ID. The program should handle errors such as invalid station codes, unavailable services, and malformed API responses. Display the results in a clean, readable table showing the route number, destination, and arrival time. Keep the code simple, well-commented, and beginner-friendly.



#CODE:

```python
import pandas as pd
import time
import random
from datetime import datetime, timedelta

def get_mock_arrival_data(station_id: str):
    """
    Simulates fetching public transport arrival data from an API.
    Returns mock data for given station_id, simulating errors for certain IDs.
    """
    # Simulate network delay
    time.sleep(random.uniform(0.5, 1.5))

    if station_id == "INVALID123":
        return {"status": "error", "message": "Invalid Station ID. Please check the station code.", "code": 404}
    elif station_id == "NO_SERVICE":
        return {"status": "error", "message": "No service available at this station currently.", "code": 503}
    elif station_id == "MALFORMED":
        # Simulate a malformed API response
        return {"status": "success", "data": "This is not valid JSON data for arrivals"}
    else:
        # Generate mock arrival data for a valid station
        arrivals = []
        current_time = datetime.now()
        for i in range(5):
            route_num = f"R{random.randint(1, 99)}"
            destination = random.choice(["Downtown", "Uptown", "Airport", "Central Station", "Suburbia"])
            # Generate arrival times in the near future
            arrival_time = current_time + timedelta(minutes=random.randint(1, 30))
            arrivals.append({
                "route_number": route_num,
                "destination": destination,
                "scheduled_arrival": arrival_time.strftime("%Y-%m-%dT%H:%M:%S")
            })
        return {"status": "success", "data": arrivals}

print("Mock API data function defined.")
```

```
Mock API data function defined.
```

2. Parse and Display Arrival Data

This function will take the data received from our mock API and display it in a user-friendly table. It includes error handling for the different scenarios we defined in the mock API.

```python
def display_arrivals(api_response):
    """
    Parses the API response and displays the next 5 arrivals in a table.
```

```
if not isinstance(api_response, dict) or "status" not in api_response:
    print("Error: Malformed API response structure. Expected dictionary with 'status'.")
    return

if api_response["status"] == "error":
    message = api_response.get("message", "An unknown error occurred.")
    code = api_response.get("code", "N/A")
    print(f"API Error (Code: {code}): {message}")
    return

if api_response["status"] == "success":
    data = api_response.get("data")
    if not isinstance(data, list):
        print("Error: Malformed API response structure. 'data' field is not a list.")
        return

    if not data:
        print("No upcoming arrivals found for this station.")
        return

    # Prepare data for display
    parsed_arrivals = []
    for arrival in data:
        try:
            route = arrival.get("route_number", "N/A")
```

```
                destination = arrival.get("destination", "N/A")
                arrival_time_str = arrival.get("scheduled_arrival", "N/A")

                # Format arrival time nicely
                if arrival_time_str != "N/A":
                    arrival_dt = datetime.strptime(arrival_time_str, "%Y-%m-%dT%H:%M:%S")
                    formatted_time = arrival_dt.strftime("%H:%M:%S") # Example: 14:35:01
                    # Calculate time until arrival
                    time_until = arrival_dt - datetime.now()
                    minutes_until = int(time_until.total_seconds() / 60)
                    if minutes_until < 1:
                        display_time = "Due"
                    elif minutes_until == 1:
                        display_time = "1 min"
                    else:
                        display_time = f"{minutes_until} mins"
                else:
                    formatted_time = "N/A"
                    display_time = "N/A"

                parsed_arrivals.append({
                    "Route": route,
                    "Destination": destination,
                    "Arrival Time": formatted_time,
                    "Time Until": display_time
```

```
                })
            except Exception as e:
                print(f"Warning: Could not parse an arrival entry: {arrival}. Error: {e}")
                continue

        if parsed_arrivals:
            df = pd.DataFrame(parsed_arrivals)
            print("\n--- Next 5 Arrivals ---")
            display(df.head(5))
        else:
            print("No valid arrival entries could be parsed.")

print("Display arrivals function defined.")
```

```
Display arrivals function defined.
```

```
# --- Main program execution ---

if __name__ == "__main__":
    # Ask the user for a station ID
    user_station_id = input("Enter the station or stop ID (e.g., TEST123, INVALID123, NO_SERVICE, MALFORMED): ")

    if not user_station_id.strip():
        print("Station ID cannot be empty. Please try again.")
    else:
        print(f"Fetching arrival data for station ID: '{user_station_id}'...")
        # Call the mock API
        response = get_mock_arrival_data(user_station_id)

        # Display the results
        display_arrivals(response)
```

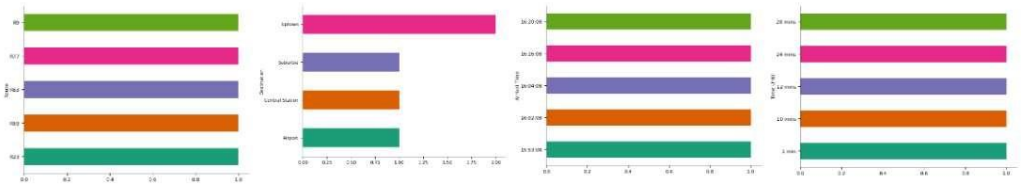
#OUTPUT:

```
Enter the station or stop ID (e.g., TEST123, INVALID123, NO_SERVICE, MALFORMED): TEST123
Fetching arrival data for station ID: 'TEST123'...

--- Next 5 Arrivals ---
```

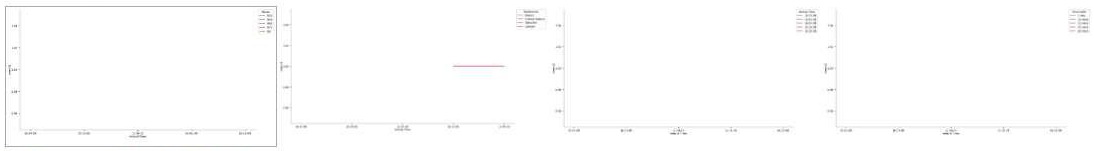
	Route	Destination	Arrival Time	Time Until
0	R23	Uptown	16:04:08	12 mins
1	R63	Central Station	16:20:08	28 mins
2	R59	Suburbia	15:53:08	1 min
3	R77	Uptown	16:02:08	10 mins
4	R9	Airport	16:16:08	24 mins

Categorical distributions

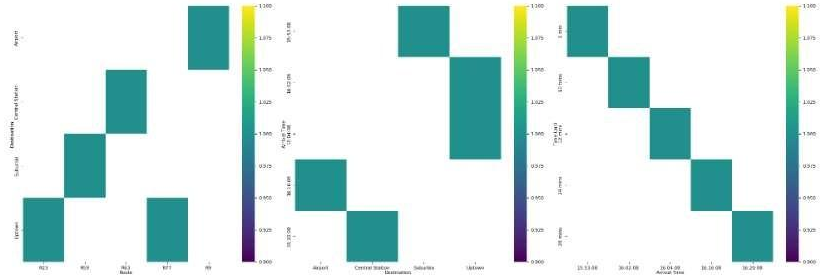


Time series

Time series



2-d categorical distributions



TASK DESCRIPTION-3:

Stock Market/Financial Data API

Task: Connect to a stock data API (e.g., Alpha Vantage, Yahoo Finance) to fetch daily stock prices.

Instructions:

Prompt AI to generate Python function to query stock data by ticker symbol.

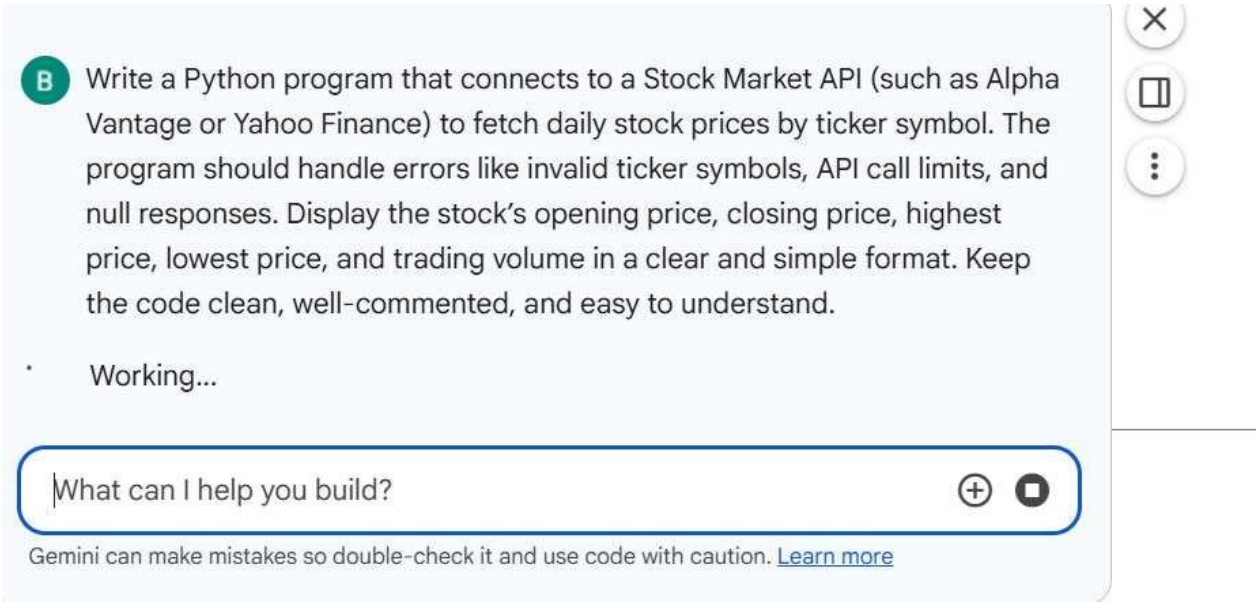
Handle API call limits, invalid ticker symbols, and null responses.

Display opening price, closing price, high, low, and trading volume. Give me perfect and simple prompt for the question

#PROMPT:

Write a Python program that connects to a Stock Market API (such as Alpha Vantage or Yahoo Finance) to fetch daily stock prices by ticker symbol. The program should handle errors like invalid ticker symbols,

API call limits, and null responses. Display the stock's opening price, closing price, highest price, lowest price, and trading volume in a clear and simple format. Keep the code clean, well-commented, and easy to understand.



#CODE:

```
# Import necessary libraries
import requests
from google.colab import userdata
import pandas as pd # For potential future data handling

# Securely retrieve the Alpha Vantage API key from Colab secrets
ALPHA_VANTAGE_API_KEY = userdata.get('ALPHA_VANTAGE_API_KEY')

# Check if the API key is available
if not ALPHA_VANTAGE_API_KEY:
    raise ValueError("ALPHA_VANTAGE_API_KEY not found in Colab secrets. Please add it as instructed above.")

print("Alpha Vantage API Key loaded successfully.")
```

```
Alpha Vantage API Key loaded successfully.
```

```
ALPHA_VANTAGE_API_KEY = "F4KERRBBSI8459QU"
```

```python
def get_stock_prices(ticker_symbol: str, api_key: str):
    """
    Fetches daily stock prices from Alpha Vantage for a given ticker symbol.

    Args:
        ticker_symbol (str): The ticker symbol of the stock (e.g., 'AAPL', 'GOOGL').
        api_key (str): Your Alpha Vantage API key.

    Returns:
        dict: A dictionary containing the latest daily stock data if successful, otherwise None.
    """
    base_url = "https://www.alphavantage.co/query"
    params = {
        'function': 'TIME_SERIES_DAILY_ADJUSTED',
        'symbol': ticker_symbol,
        'apikey': api_key,
        'outputsize': 'compact' # Get only the most recent 100 data points
    }

    try:
        response = requests.get(base_url, params=params, timeout=15)
        response.raise_for_status() # Raise an HTTPError for bad responses (4xx or 5xx)

        data = response.json()

    # Handle Alpha Vantage specific error messages
    if "Error Message" in data:
        print(f"Error: {data['Error Message']} for ticker '{ticker_symbol}'. Please check the ticker symbol or your API key.")
        return None
    if "Note" in data:
        print(f"API Note: {data['Note']} for ticker '{ticker_symbol}'. This might indicate API call limits.")
        # Depending on the note, you might still get data or need to handle it as an error
        # For now, we'll proceed if there's still time series data

    time_series = data.get('Time Series (Daily)')

    if not time_series:
        print(f"Error: No daily time series data found for '{ticker_symbol}'. Invalid ticker or no data available.")
        return None

    # Get the latest date available in the time series
    latest_date = sorted(time_series.keys(), reverse=True)[0]
    latest_data = time_series[latest_date]

    # Extract and convert to float (or appropriate type)
    stock_details = {
        'Date': latest_date,
        'Open': float(latest_data['1. open']),
        'High': float(latest_data['2. high']),
        'Low': float(latest_data['3. low'])
```

```
            'Close': float(latest_data['4. close']),
            'Volume': int(latest_data['6. volume'])
        }
        return stock_details

    except requests.exceptions.Timeout:
        print("Error: Request timed out. Please check your internet connection or try again later.")
        return None
    except requests.exceptions.ConnectionError:
        print("Error: Could not connect to the Alpha Vantage API. Check your internet connection.")
        return None
    except requests.exceptions.HTTPError as err:
        print(f"Error: HTTP error occurred: {err}")
        return None
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        return None

def display_stock_data(stock_data: dict):
```

```
    if stock_data:
        print(f"\n--- Stock Details for {stock_data.get('Date')} ---")
        print(f"  Open:  {stock_data.get('Open'):.2f}")
        print(f"  High:  {stock_data.get('High'):.2f}")
        print(f"  Low:   {stock_data.get('Low'):.2f}")
        print(f"  Close: {stock_data.get('Close'):.2f}")
        print(f"  Volume: {stock_data.get('Volume'):,}")
    else:
        print("No stock data to display.")

print("Stock price functions defined.")
```

```
Stock price functions defined.
```

```
# --- Main program execution ---

if __name__ == "__main__":
    user_ticker = input("Enter the stock ticker symbol (e.g., AAPL, IBM): ")

    if not user_ticker.strip():
        print("Ticker symbol cannot be empty. Please try again.")
    else:
        print(f"Fetching daily stock prices for '{user_ticker.upper()}'...")
        stock_info = get_stock_prices(user_ticker.upper(), ALPHA_VANTAGE_API_KEY)

        display_stock_data(stock_info)
```

#OUTPUT:

```
Enter the stock ticker symbol (e.g., AAPL, IBM): IBM
Fetching daily stock prices for 'IBM'...
Error: No daily time series data found for 'IBM'. Invalid ticker or no data available.
No stock data to display.
```

TASK DISCRIPTION-4:

Real-Time Application: Translation API

Scenario: Build a translator using a free Translation API (e.g., Libre Translate, Google Translate).

Requirements:

Accept input text and target language from the user.

Handle invalid language codes, API quota exceeded, and empty text input.

Display original and translated text clearly.

Implement a retry mechanism if the API fails on the first attempt.

#PROMPT:

Write a Python program that uses a free Translation API (like Libre Translate or Google Translate) to build a simple translator. The program should take text and a target language from the user, then display both the original and translated text clearly. Handle errors such as invalid language codes, empty input, and API quota exceeded. Include a retry mechanism to attempt the translation again if the API request fails the first time. Keep the code clean, well-commented, and easy to understand.

#CODE:

```python
import requests
import time

def translate_text(text: str, target_language: str, source_language: str = 'auto', retries: int = 3):
    """
    Translates text using a public LibreTranslate API instance.

    Args:
        text (str): The text to be translated.
        target_language (str): The target language code (e.g., 'es', 'fr', 'de').
        source_language (str): The source language code (or 'auto' for auto-detection).
        retries (int): Number of times to retry the API call if it fails.

    Returns:
        str: The translated text, or an error message if translation fails.
    """
    if not text.strip():
        return "Error: Input text cannot be empty."

    if not target_language.strip():
        return "Error: Target language code cannot be empty."
```

```python
# Using a public LibreTranslate instance. Be aware of rate limits and availability.
# For production, consider self-hosting LibreTranslate or using a dedicated API key service.
api_url = "https://translate.argosopentech.com/translate"

headers = {
    "Content-Type": "application/json"
}
payload = {
    "q": text,
    "source": source_language,
    "target": target_language
}

for attempt in range(retries):
    try:
        response = requests.post(api_url, headers=headers, json=payload, timeout=10)
        response.raise_for_status() # Raise HTTPError for bad responses (4xx or 5xx)

        translation_data = response.json()
        translated_text = translation_data.get('translatedText')

        if translated_text:
            return translated_text
        else:
            # LibreTranslate might return errors in a different format
```

```python
                # LibreTranslate might return errors in a different format
                error_message = translation_data.get('error', 'Unknown translation error.')
                return f"Translation API Error: {error_message}"

            except requests.exceptions.Timeout:
                print(f"Attempt {attempt + 1}/{retries}: Request timed out. Retrying...")
            except requests.exceptions.ConnectionError:
                print(f"Attempt {attempt + 1}/{retries}: Could not connect to the translation API. Retrying...")
            except requests.exceptions.HTTPError as err:
                if err.response.status_code == 400:
                    # Specific handling for 400 Bad Request which often means invalid language
                    print(f"Attempt {attempt + 1}/{retries}: Invalid language code or bad request (HTTP 400). Please check your language codes.
                elif err.response.status_code == 429:
                    print(f"Attempt {attempt + 1}/{retries}: API quota exceeded or too many requests (HTTP 429). Retrying after a delay...")
                    time.sleep(2 ** attempt) # Exponential backoff
                else:
                    print(f"Attempt {attempt + 1}/{retries}: HTTP error occurred: {err}. Retrying...")
            except Exception as e:
                print(f"Attempt {attempt + 1}/{retries}: An unexpected error occurred: {e}. Retrying...")

            if attempt < retries - 1:
                time.sleep(1) # Short delay before next retry for non-429 errors

        return "Error: Translation failed after multiple retries. Please try again later or check your input."
```

```python
# --- Main program execution ---

if __name__ == "__main__":
    print("\n--- Simple Text Translator ---")
    user_text = input("Enter the text to translate: ")
    user_target_language = input("Enter the target language code (e.g., es, fr, de): ")

    if not user_text.strip() or not user_target_language.strip():
        print("\nError: Both text and target language must be provided.")
    else:
        print(f"\nTranslating '{user_text}' to '{user_target_language}'...")
        translated = translate_text(user_text, user_target_language)

        print("\n--- Translation Result ---")
        print(f"Original: {user_text}")
        print(f"Translated: {translated}")
```

#OUTPUT:

```
--- Simple Text Translator ---
Enter the text to translate: Hello world!
Enter the target language code (e.g., es, fr, de): fr

Translating 'Hello world!' to 'fr'...
Attempt 1/3: Could not connect to the translation API. Retrying...
Attempt 2/3: Could not connect to the translation API. Retrying...
Attempt 3/3: Could not connect to the translation API. Retrying...

--- Translation Result ---
Original: Hello world!
Translated: Error: Translation failed after multiple retries. Please try again later or check your input.
```

Thank You