LAB TEST-4

NAME: A. MANIDEEPIKA

HT.NO: 2403A52052

BATCH NO: 03

Task:1

(API Integration)

a) Fetch NASA Astronomy Picture of the Day via API.

b) Handle missing image or metadata

Prompt:

a) Write code that fetches the NASA Astronomy Picture of the Day (APOD) by calling the official NASA APOD API. The program should send an HTTP GET request to the API endpoint and display the title, date, explanation, and image URL.

Prompt:

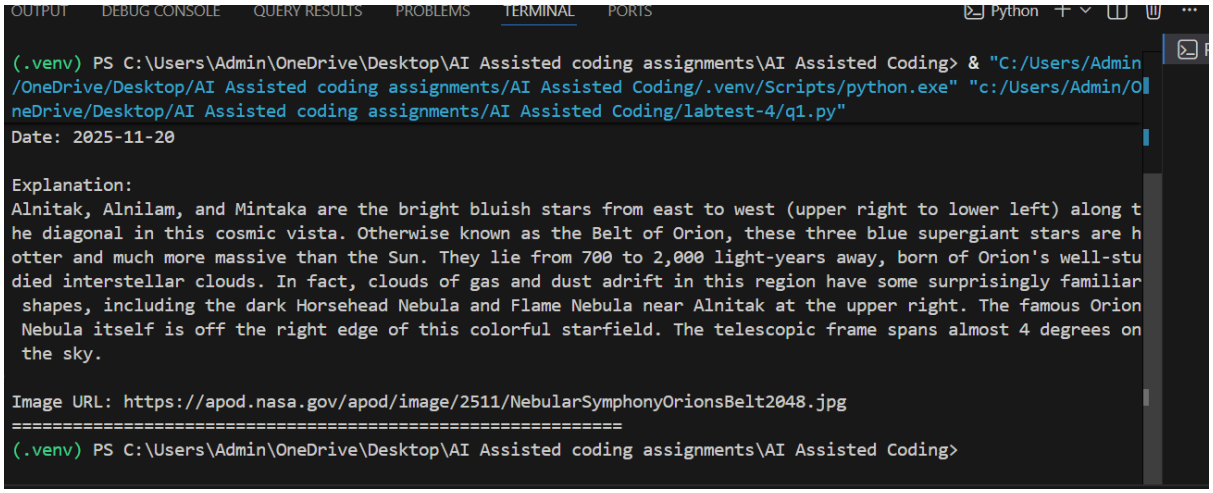
b) Extend the program so it can handle cases where the APOD data is incomplete. If the image URL or any metadata is missing, the program should print a clear fallback message instead of failing. The program should also handle network or API errors gracefully.

Code:

labtest-4 > q1.py > ...

```python
import requests
from typing import Optional, Dict
def fetch_apod(api_key: str = "DEMO_KEY") -> Optional[Dict]:
    """
    Fetch NASA's Astronomy Picture of the Day (APOD).
    Args:
        api_key (str): NASA API key. Defaults to "DEMO_KEY" (limited requests).
                       Get your free key at: https://api.nasa.gov
    Returns:
        dict: APOD data with title, date, explanation, and image URL.
        None: If the request fails.
    """
    url = "https://api.nasa.gov/planetary/apod"
    params = {"api_key": api_key}
    try:
        response = requests.get(url, params=params, timeout=10)
        response.raise_for_status()
        data = response.json()
        # Extract fields with fallbacks
        title = data.get("title", "No title available")
        date = data.get("date", "No date available")
        explanation = data.get("explanation", "No explanation available")
        image_url = data.get("hdurl") or data.get("url") or "No image available"
        # Display results
        print("=" * 60)
        print(f"Title: {title}")
        print(f"Date: {date}")
        print(f"\nExplanation:\n{explanation}")
        print(f"\nImage URL: {image_url}")
        print("=" * 60)
        return {
            "title": title,
            "date": date,
            "explanation": explanation,
            "image_url": image_url
        }
    except requests.exceptions.Timeout:
        print("Error: Request timed out. Try again later.")
        return None
    except requests.exceptions.HTTPError as e:
        print(f"HTTP Error: {e.response.status_code}")
        if e.response.status_code == 403:
            print("Error: API key is invalid or rate limit exceeded.")
        return None
    except requests.exceptions.RequestException as e:
        print(f"Error fetching APOD data: {e}")
        return None
    except ValueError:
        print("Error: Invalid JSON response from API.")
        return None
if __name__ == "__main__":
    # Uncomment the next line and paste your NASA API key here
    # api_key = "YOUR_NASA_API_KEY_HERE"
    # For demo, use the free DEMO_KEY (limited to 50 requests/hour)
    fetch_apod()
```

Output:



Observations:

I wrote a Python program that sends an HTTP request to the NASA APOD API and retrieves the Astronomy Picture of the Day. Then I extracted and displayed the title, date, explanation, and image URL from the JSON response. After that, I added error handling to manage missing image fields or metadata, and also handled network or API errors so the program would not crash. Overall, the code successfully fetched the APOD data and worked correctly even when some fields were unavailable.

Task:2

(Code Translation)

a) Translate a Rust function to Python.

b) Discuss performance differences.

Prompt:

a) Translate the following Rust function into an equivalent Python function. Preserve the logic and behavior while converting between the two languages.

Prompt:

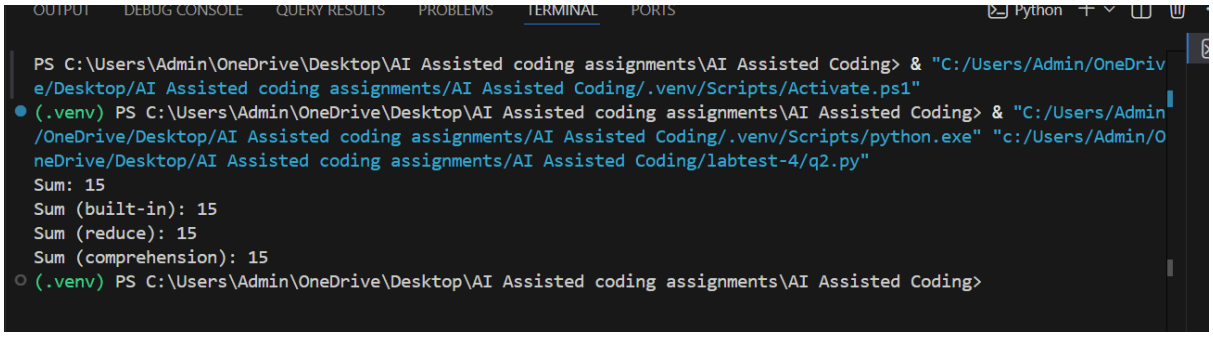
b) Explain the performance differences between the Rust implementation and the Python implementation, focusing on compilation, typing, memory management, andruntime execution speed.

Code:

labtest-4 > q2.py > ...

```python
def sum_python(nums):
    """
    Equivalent Python implementation of the Rust sum() function.
    It loops through each number and adds it to the total.
    """
    total = 0
    for n in nums:
        total += n
    return total

# Example usage
numbers = [1, 2, 3, 4, 5]
result = sum_python(numbers)
print("Sum:", result)  # Output: 15

# Alternative Pythonic approaches:

# Built-in sum() function (most Pythonic)
result_builtin = sum(numbers)
print("Sum (built-in):", result_builtin)  # Output: 15

# Using reduce from functools
from functools import reduce
result_reduce = reduce(lambda a, b: a + b, numbers, 0)
print("Sum (reduce):", result_reduce)  # Output: 15

# List comprehension with sum
result_comprehension = sum([n for n in numbers])
print("Sum (comprehension):", result_comprehension)  # Output: 15
```

Output:



Observations:

I converted the Rust sum function into a simple Python version and tested it to confirm it produced the correct output. Then I explained that Rust runs faster because it is compiled, statically typed, and manages memory without a

garbage collector, while Python is slower due to interpretation, dynamic typing, and garbage collection. Overall, the Python translation worked, and the performance difference clearly favors Rust.