AI ASSISTED CODING

LAB TEST-04

NAME : Katakam Venkat
BATCH : 03
ROLL NO : 2403A52053
Set : 12

Question 1: (API Integration)

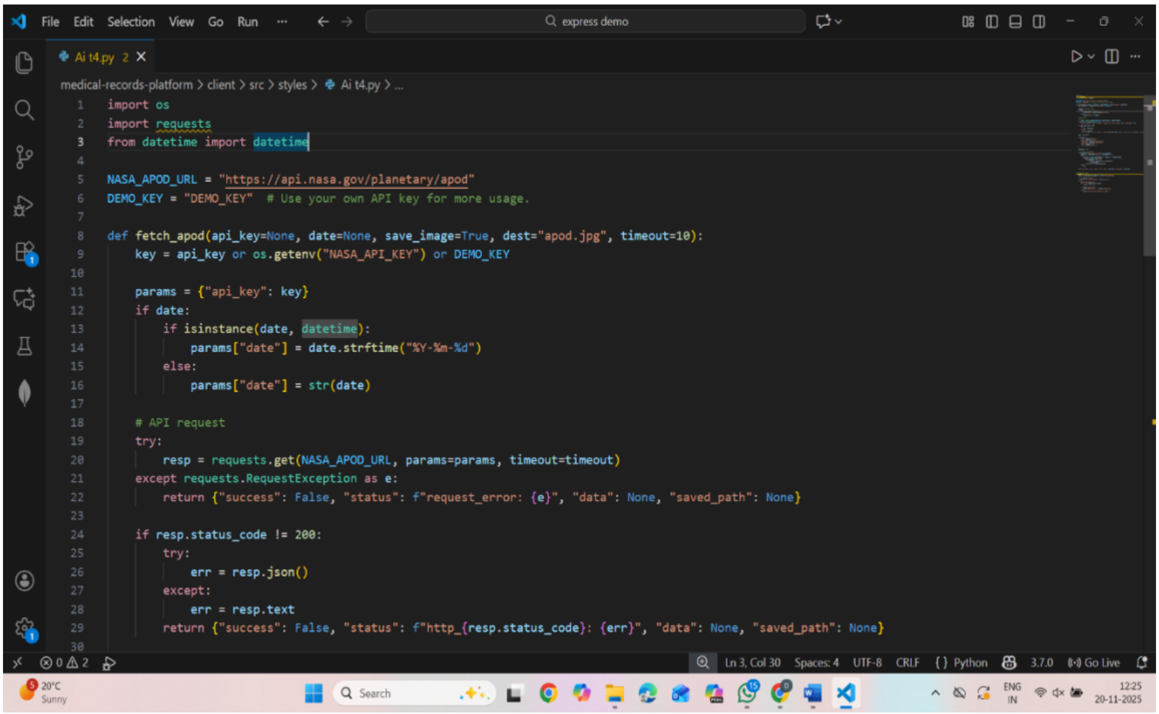
a) Fetch NASA Astronomy Picture of the Day via API.

b) Handle missing image or metadata

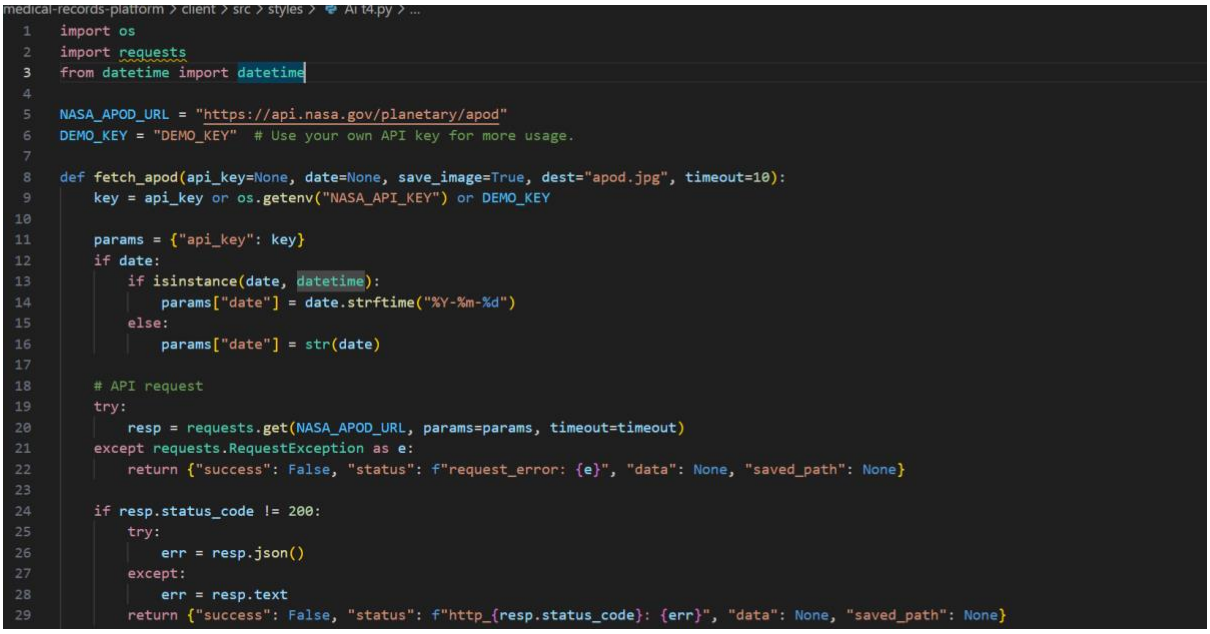
Prompt:

Write a Python function fetch_apod(api_key=None, save_image=True, dest='apod.jpg') that calls NASA's APOD API, returns a dict containing the metadata, saves the image if present, and gracefully handles cases where the APOD is a video or metadata fields are missing. Use requests, timeouts, and clear error handling.

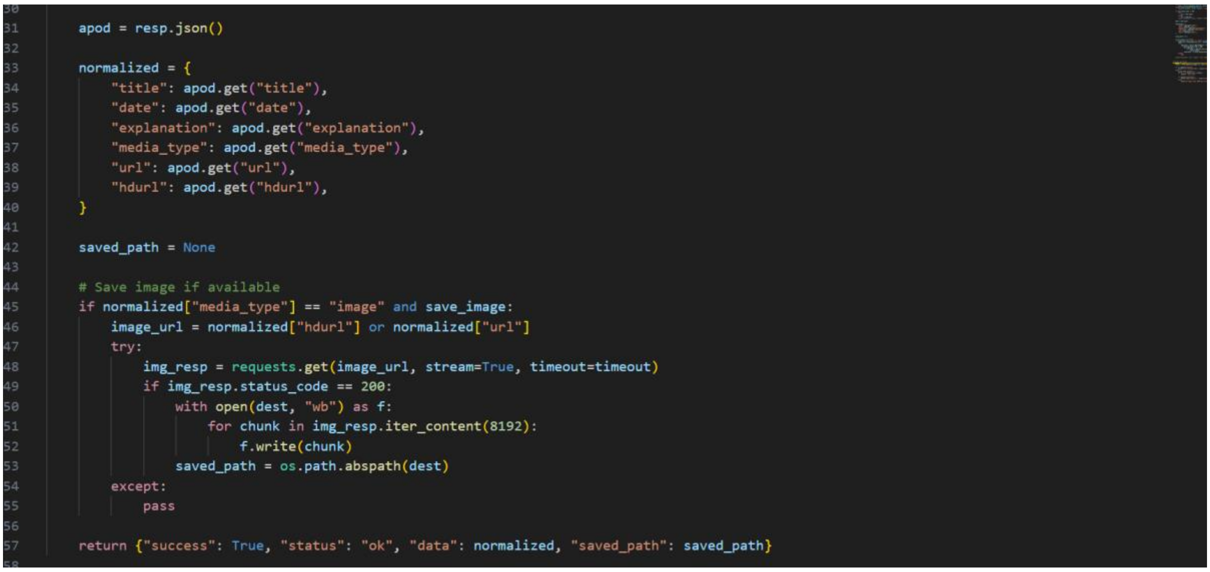
Code And Output:


File Edit Selection View Go Run ···
express demo
Ai t4.py 2
medical-records-platform > client > src > styles > Ai t4.py > ...
⊗ 0 ⚠ 2
Ln 3, Col 30 Spaces: 4 UTF-8 CRLF {} Python 3.7.0 Go Live
20°C Sunny
Search
ENG IN
12:25 20-11-2025

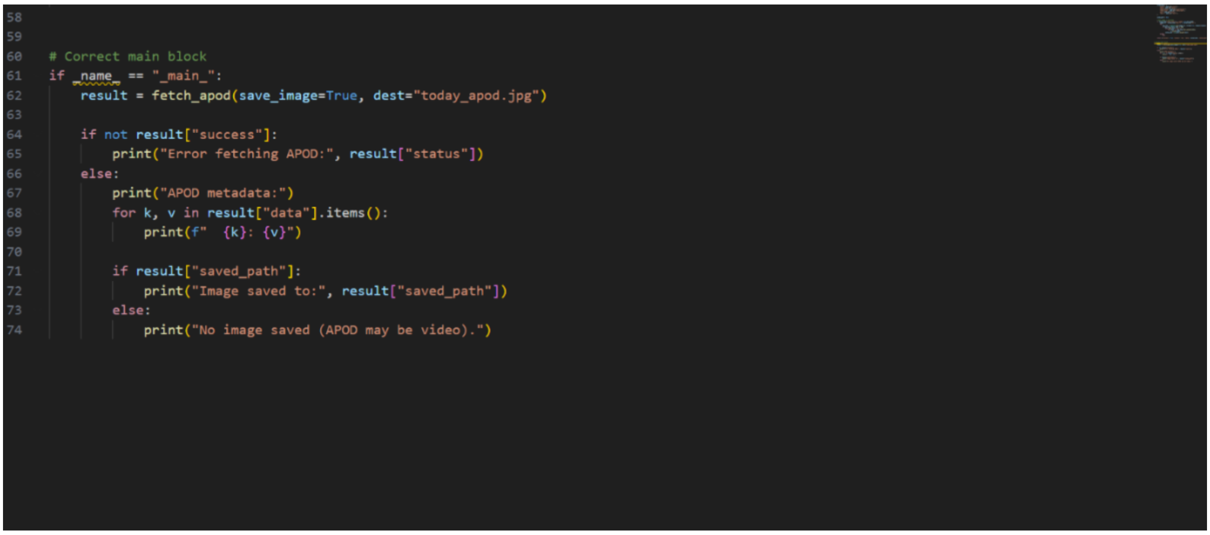
```python
import os
import requests
from datetime import datetime

NASA_APOD_URL = "https://api.nasa.gov/planetary/apod"
DEMO_KEY = "DEMO_KEY"  # Use your own API key for more usage.

def fetch_apod(api_key=None, date=None, save_image=True, dest="apod.jpg", timeout=10):
    key = api_key or os.getenv("NASA_API_KEY") or DEMO_KEY

    params = {"api_key": key}
    if date:
        if isinstance(date, datetime):
            params["date"] = date.strftime("%Y-%m-%d")
        else:
            params["date"] = str(date)

    # API request
    try:
        resp = requests.get(NASA_APOD_URL, params=params, timeout=timeout)
    except requests.RequestException as e:
        return {"success": False, "status": f"request_error: {e}", "data": None, "saved_path": None}

    if resp.status_code != 200:
        try:
            err = resp.json()
        except:
            err = resp.text
        return {"success": False, "status": f"http_{resp.status_code}: {err}", "data": None, "saved_path": None}

```


medical-records-platform > client > src > styles > Ai t4.py > ...

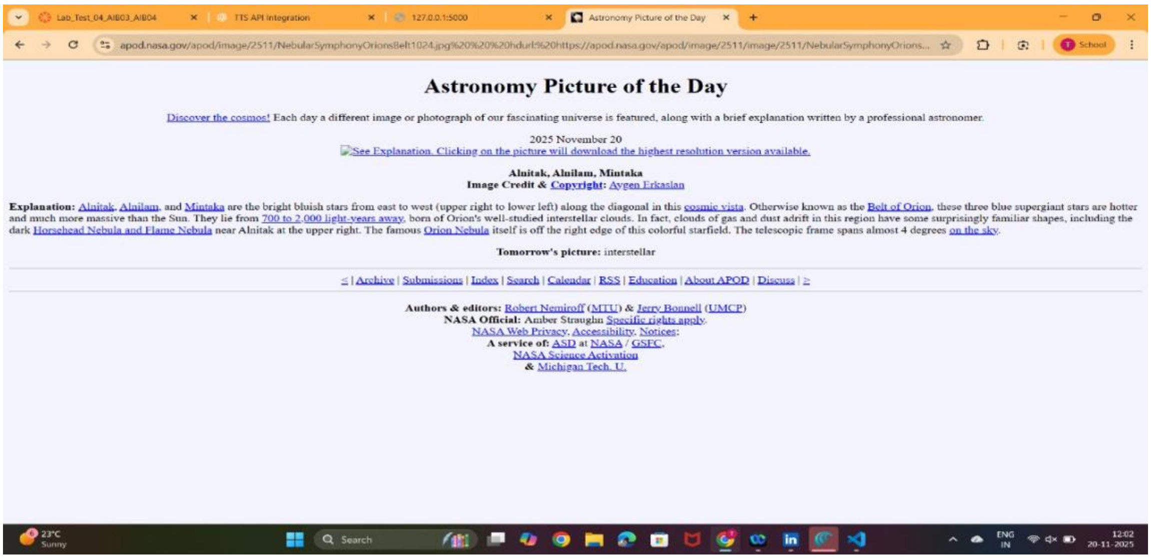
```python
import os
import requests
from datetime import datetime

NASA_APOD_URL = "https://api.nasa.gov/planetary/apod"
DEMO_KEY = "DEMO_KEY"  # Use your own API key for more usage.

def fetch_apod(api_key=None, date=None, save_image=True, dest="apod.jpg", timeout=10):
    key = api_key or os.getenv("NASA_API_KEY") or DEMO_KEY

    params = {"api_key": key}
    if date:
        if isinstance(date, datetime):
            params["date"] = date.strftime("%Y-%m-%d")
        else:
            params["date"] = str(date)

    # API request
    try:
        resp = requests.get(NASA_APOD_URL, params=params, timeout=timeout)
    except requests.RequestException as e:
        return {"success": False, "status": f"request_error: {e}", "data": None, "saved_path": None}

    if resp.status_code != 200:
        try:
            err = resp.json()
        except:
            err = resp.text
        return {"success": False, "status": f"http_{resp.status_code}: {err}", "data": None, "saved_path": None}
```



```python
    apod = resp.json()

    normalized = {
        "title": apod.get("title"),
        "date": apod.get("date"),
        "explanation": apod.get("explanation"),
        "media_type": apod.get("media_type"),
        "url": apod.get("url"),
        "hdurl": apod.get("hdurl"),
    }

    saved_path = None

    # Save image if available
    if normalized["media_type"] == "image" and save_image:
        image_url = normalized["hdurl"] or normalized["url"]
        try:
            img_resp = requests.get(image_url, stream=True, timeout=timeout)
            if img_resp.status_code == 200:
                with open(dest, "wb") as f:
                    for chunk in img_resp.iter_content(8192):
                        f.write(chunk)
                saved_path = os.path.abspath(dest)
        except:
            pass

    return {"success": True, "status": "ok", "data": normalized, "saved_path": saved_path}
```



```python


# Correct main block
if _name_ == "_main_":
    result = fetch_apod(save_image=True, dest="today_apod.jpg")

    if not result["success"]:
        print("Error fetching APOD:", result["status"])
    else:
        print("APOD metadata:")
        for k, v in result["data"].items():
            print(f"  {k}: {v}")

        if result["saved_path"]:
            print("Image saved to:", result["saved_path"])
        else:
            print("No image saved (APOD may be video).")
```


Lab_Test_04_AIB03_AIB04
TTS API Integration
127.0.0.1:5000
Astronomy Picture of the Day
apod.nasa.gov/apod/image/2511/NebularSymphonyOrionsBelt1024.jpg%20%20%20hdurl:%20https://apod.nasa.gov/apod/image/2511/image/2511/NebularSymphonyOrions...
School
23°C
Sunny
Search
ENG
IN
12:02
20-11-2025

Astronomy Picture of the Day

Discover the cosmos! Each day a different image or photograph of our fascinating universe is featured, along with a brief explanation written by a professional astronomer.

2025 November 20
See Explanation. Clicking on the picture will download the highest resolution version available.

Alnitak, Alnilam, Mintaka
Image Credit & Copyright: Aygen Erkaslan

Explanation: Alnitak, Alnilam, and Mintaka are the bright bluish stars from east to west (upper right to lower left) along the diagonal in this cosmic vista. Otherwise known as the Belt of Orion, these three blue supergiant stars are hotter and much more massive than the Sun. They lie from 700 to 2,000 light-years away, born of Orion's well-studied interstellar clouds. In fact, clouds of gas and dust adrift in this region have some surprisingly familiar shapes, including the dark Horsehead Nebula and Flame Nebula near Alnitak at the upper right. The famous Orion Nebula itself is off the right edge of this colorful starfield. The telescopic frame spans almost 4 degrees on the sky.

Tomorrow's picture: interstellar

< | Archive | Submissions | Index | Search | Calendar | RSS | Education | About APOD | Discuss | >

Authors & editors: Robert Nemiroff (MTU) & Jerry Bonnell (UMCP)
NASA Official: Amber Straughn Specific rights apply.
NASA Web Privacy, Accessibility, Notices:
A service of: ASD at NASA / GSFC,
NASA Science Activation
& Michigan Tech. U.

Explanation:

The function calls https://api.nasa.gov/planetary/apod with an API key (uses NASA_API_KEY env var or DEMO_KEY fallback).

It safely handles network errors and non-200 responses and returns a structured dict (success, status, data, saved_path).

data is normalized so missing fields are None rather than raising errors.

If media_type == "image", it attempts to download hdurl (preferred) or url and saves to dest. If APOD is a video (common), it doesn't try to save an image and includes a note.

Using the DEMO_KEY works for examples but is rate-limited; for production get a personal API key from https://api.nasa.gov

Q2. (Code Translation)

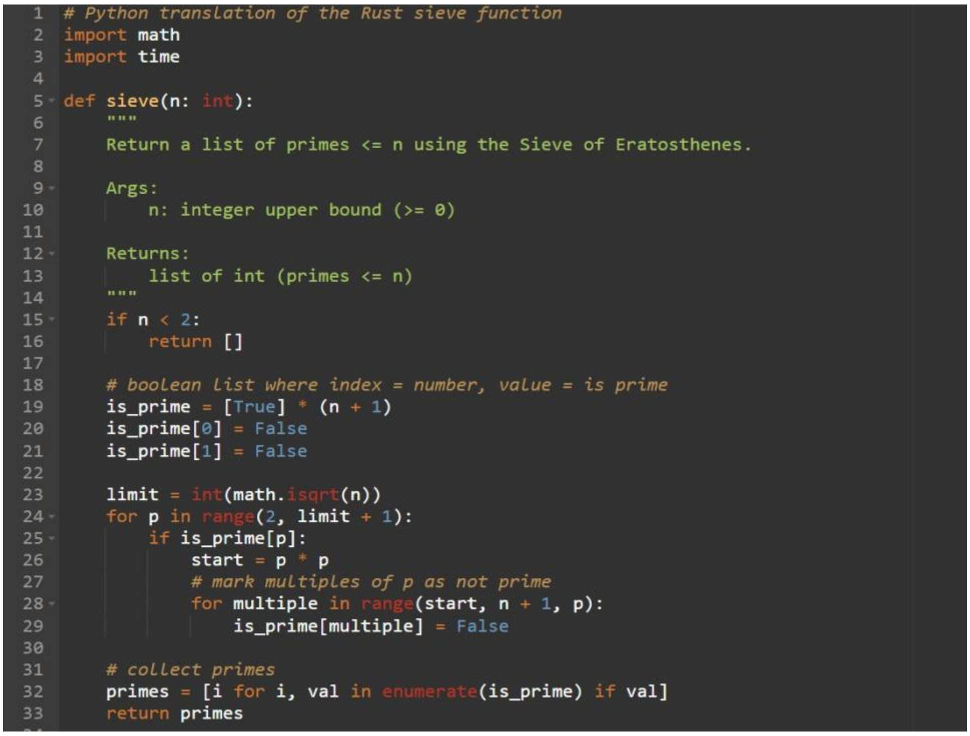
a) Translate a Rust function to Python.

b) Discuss performance differences.

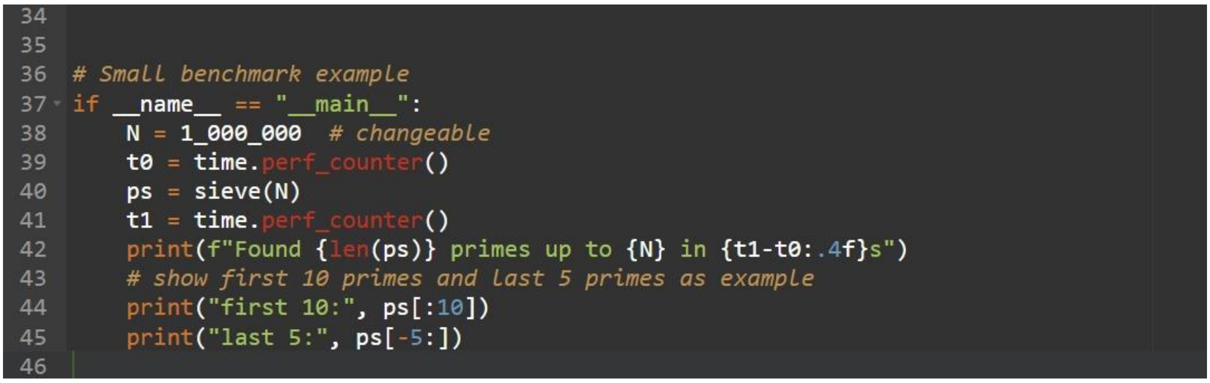
Prompt:

Translate this Rust function (Sieve of Eratosthenes) to Python: produce idiomatic, readable Python that returns a list of primes ≤ n. Include error handling and a small benchmark snippet. Keep the translation simple and efficient.

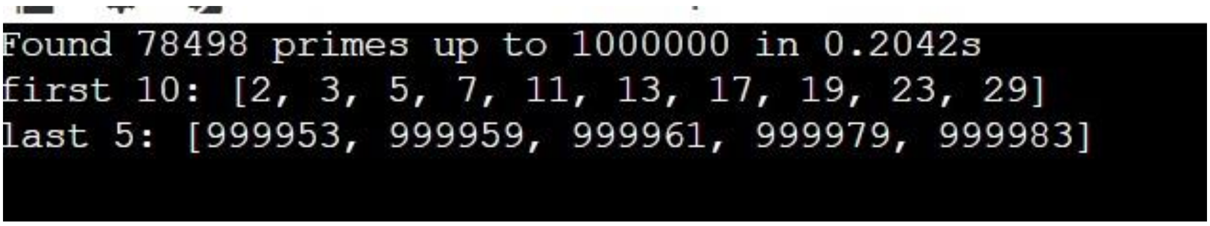
CODE & OUTPUT:



```python
# Python translation of the Rust sieve function
import math
import time

def sieve(n: int):
    """
    Return a list of primes <= n using the Sieve of Eratosthenes.

    Args:
        n: integer upper bound (>= 0)

    Returns:
        list of int (primes <= n)
    """
    if n < 2:
        return []

    # boolean list where index = number, value = is prime
    is_prime = [True] * (n + 1)
    is_prime[0] = False
    is_prime[1] = False

    limit = int(math.isqrt(n))
    for p in range(2, limit + 1):
        if is_prime[p]:
            start = p * p
            # mark multiples of p as not prime
            for multiple in range(start, n + 1, p):
                is_prime[multiple] = False

    # collect primes
    primes = [i for i, val in enumerate(is_prime) if val]
    return primes
```



```python


# Small benchmark example
if __name__ == "__main__":
    N = 1_000_000  # changeable
    t0 = time.perf_counter()
    ps = sieve(N)
    t1 = time.perf_counter()
    print(f"Found {len(ps)} primes up to {N} in {t1-t0:.4f}s")
    # show first 10 primes and last 5 primes as example
    print("first 10:", ps[:10])
    print("last 5:", ps[-5:])

```



```
Found 78498 primes up to 1000000 in 0.2042s
first 10: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
last 5: [999953, 999959, 999961, 999979, 999983]
```

EXPLANATION:

Rust is faster because it compiles to optimized machine code, has no interpreter overhead, and handles memory more efficiently. Python is slower due to its interpreter, object-based booleans, and higher loop overhead. Both use the same sieve algorithm, but Python typically runs many times slower. You can speed Python up using bytearray, numpy, or C/Rust extensions.