```
import requests
import numpy
import matplotlib
import ipywidgets
!pip install skyfield
```

```
Collecting skyfield
  Downloading skyfield-1.53-py3-none-any.whl.metadata (2.4 kB)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from skyfield) (2025.7.14)
Collecting jplephem>=2.13 (from skyfield)
  Downloading jplephem-2.23-py3-none-any.whl.metadata (23 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from skyfield) (2.0.2)
Collecting sgp4>=2.13 (from skyfield)
  Downloading sgp4-2.24-cp311-cp311-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
Downloading skyfield-1.53-py3-none-any.whl (366 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 367.0/367.0 kB 16.0 MB/s eta 0:00:00
Downloading jplephem-2.23-py3-none-any.whl (49 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.4/49.4 kB 3.6 MB/s eta 0:00:00
Downloading sgp4-2.24-cp311-cp311-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (234 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 234.5/234.5 kB 16.7 MB/s eta 0:00:00
Installing collected packages: sgp4, jplephem, skyfield
Successfully installed jplephem-2.23 sgp4-2.24 skyfield-1.53
```

Task 1

```
import skyfield
```

```
from datetime import datetime, timedelta

def fetch_iss_tle():
    tle_url = "https://celestrak.org/NORAD/elements/stations.txt"

    try:
        # Make a GET request to the TLE endpoint
        response = requests.get(tle_url)
        response.raise_for_status()  # Raise an error for bad status codes

        lines = response.text.strip().splitlines()

        # Parse the response to find the ISS entry
        for i in range(len(lines)):
            if "ISS (ZARYA)" in lines[i]:
                name = lines[i].strip()
                line1 = lines[i + 1].strip()
                line2 = lines[i + 2].strip()

                # Extract the epoch year and day from Line 1
                epoch_raw = line1[18:32].strip()
                epoch_year = int("20" + epoch_raw[:2])  # TLE format gives last 2 digits of year
                epoch_day = float(epoch_raw[2:])

                # Convert epoch to datetime
                epoch_datetime = datetime(epoch_year, 1, 1) + timedelta(days=epoch_day - 1)

                # Compare with current UTC time
                now = datetime.utcnow()
                time_diff = abs((now - epoch_datetime).total_seconds())

                # Allow ~24-hour window
                if time_diff <= 86400:
                    print("Successfully fetched recent TLE for ISS:")
                    print(name)
                    print(line1)
                    print(line2)
                    return
                else:
                    print("TLE data is outdated. Epoch:", epoch_datetime, "UTC Now:", now)
                    return

        print("ISS (ZARYA) TLE not found in the response.")

    except requests.exceptions.RequestException as e:
        print("HTTP Request failed:", e)
    except IndexError:
        print("TLE format error: Missing lines.")
    except Exception as e:
        print("Unexpected error:", e)
```

```
# Run the function
fetch_iss_tle()
```

```
Successfully fetched recent TLE for ISS:
ISS (ZARYA)
1 25544U 98067A   25204.25047604  .00008934  00000+0  16419-3 0  9998
2 25544  51.6351 128.3174 0002345 109.2368 347.8357 15.50024026520753
```

```
from skyfield.api import load, EarthSatellite, wgs84, Topos, utc
from datetime import datetime, timedelta
import pytz
import requests
import numpy as np # Import numpy for array handling

def get_tle_for_iss():
    url = "https://celestrak.org/NORAD/elements/stations.txt"
    try:
        response = requests.get(url)
        response.raise_for_status()
        lines = response.text.strip().splitlines()
        for i, line in enumerate(lines):
            if "ISS (ZARYA)" in line:
                return lines[i].strip(), lines[i+1].strip(), lines[i+2].strip()
        raise ValueError("ISS TLE not found.")
    except Exception as e:
        print("Failed to fetch TLE data:", e)
        exit()

def get_user_coordinates():
    try:
        lat = float(input("Enter your latitude (decimal degrees, -90 to 90): "))
        lon = float(input("Enter your longitude (decimal degrees, -180 to 180): "))
        if not (-90 <= lat <= 90 and -180 <= lon <= 180):
            raise ValueError("Invalid coordinates.")
        return lat, lon
    except ValueError:
        print("Invalid input. Please enter numbers within the valid range.")
        exit()

def find_next_pass(lat, lon, tle_lines):
    from skyfield.api import utc  # Ensure this is imported

    ts = load.timescale()
    name, line1, line2 = tle_lines
    satellite = EarthSatellite(line1, line2, name, ts)

    location = wgs84.latlon(lat, lon)

    t0 = ts.now()
    t1 = ts.utc((datetime.utcnow() + timedelta(days=1)).replace(tzinfo=utc))

    # Get all pass events (0 = rise, 1 = max, 2 = set)
    times, events = satellite.find_events(location, t0, t1, altitude_degrees=10.0)

    times = list(times)
    events = list(events)

    # Group events into full passes
    pass_events = []
    current_pass = {}

    for t, e in zip(times, events):
        if e == 0:
            current_pass = {"rise": t}
        elif e == 1 and "rise" in current_pass:
            current_pass["max"] = t
        elif e == 2 and "max" in current_pass:
            current_pass["set"] = t
            pass_events.append(current_pass)
            current_pass = {}

    if not pass_events:
        print("❌ Could not compute a full pass event within the next 24 hours.")
        return

    # Use the first full pass
    first_pass = pass_events[0]
    rise_time = first_pass["rise"]
    max_time = first_pass["max"]
    set_time = first_pass["set"]
    local_tz = pytz.timezone('Asia/Kolkata')
    rise_dt = rise_time.utc_datetime().replace(tzinfo=pytz.utc).astimezone(local_tz)
    max_dt = max_time.utc_datetime().replace(tzinfo=pytz.utc).astimezone(local_tz)
    set_dt = set_time.utc_datetime().replace(tzinfo=pytz.utc).astimezone(local_tz)

    duration = (set_dt - rise_dt).seconds
    alt, az, _ = (satellite - location).at(max_time).altaz()

    print("\n--- ✅ Next ISS Pass Details ---")
    print(f"📍 Location       : Lat {lat}, Lon {lon}")
    print(f"🔼 Rise Time      : {rise_dt.strftime('%Y-%m-%d %H:%M:%S %Z')}")
    print(f"☀️ Peak Time      : {max_dt.strftime('%Y-%m-%d %H:%M:%S %Z')}")
    print(f"🔽 Set Time       : {set_dt.strftime('%Y-%m-%d %H:%M:%S %Z')}")
    print(f"⏱ Duration       : {duration} seconds")
    print(f"📏 Peak Altitude  : {alt.degrees:.2f}°")

def main():
    print("🌍 ISS Pass Time Calculator")
    lat, lon = get_user_coordinates()
    tle_data = get_tle_for_iss()
    find_next_pass(lat, lon, tle_data)

if __name__ == "__main__":
    main()
```

```
🌍 ISS Pass Time Calculator
Enter your latitude (decimal degrees, -90 to 90): 22
Enter your longitude (decimal degrees, -180 to 180): 77

--- ✅ Next ISS Pass Details ---
📍 Location       : Lat 22.0, Lon 77.0
🔼 Rise Time      : 2025-07-24 00:06:34 IST
☀️ Peak Time      : 2025-07-24 00:09:05 IST
🔽 Set Time       : 2025-07-24 00:11:37 IST
⏱ Duration       : 302 seconds
📏 Peak Altitude  : 20.03°
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

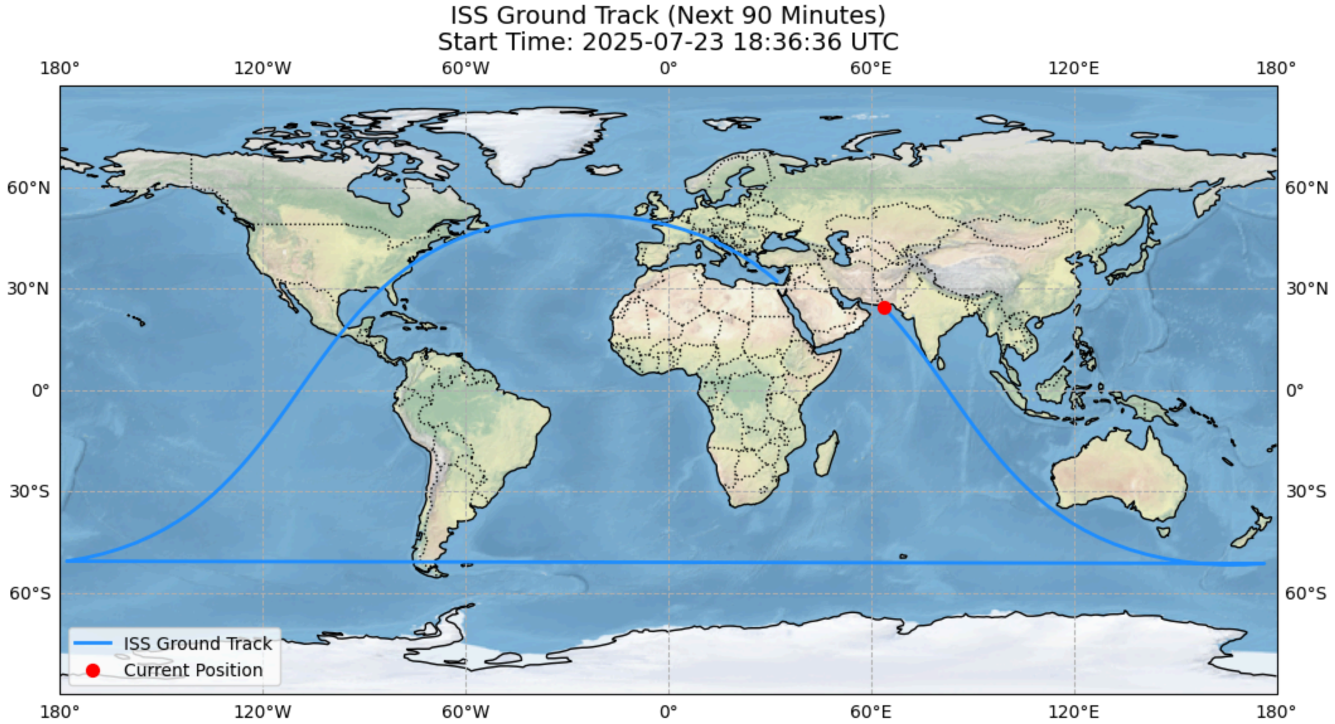
```
!pip install cartopy
```

```
Collecting cartopy
  Downloading Cartopy-0.24.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (7.9 kB)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.11/dist-packages (from cartopy) (2.0.2)
Requirement already satisfied: matplotlib>=3.6 in /usr/local/lib/python3.11/dist-packages (from cartopy) (3.10.0)
Requirement already satisfied: shapely>=1.8 in /usr/local/lib/python3.11/dist-packages (from cartopy) (2.1.1)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.11/dist-packages (from cartopy) (25.0)
Requirement already satisfied: pyshp>=2.3 in /usr/local/lib/python3.11/dist-packages (from cartopy) (2.3.1)
Requirement already satisfied: pyproj>=3.3.1 in /usr/local/lib/python3.11/dist-packages (from cartopy) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (4.59.0
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (1.4.8)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (2.9
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from pyproj>=3.3.1->cartopy) (2025.7.14)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib>=3.6->car
Downloading Cartopy-0.24.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.7/11.7 MB 93.7 MB/s eta 0:00:00
Installing collected packages: cartopy
Successfully installed cartopy-0.24.1
```

```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from skyfield.api import load, EarthSatellite, wgs84, utc
from datetime import datetime, timedelta
import requests

# --- STEP 1: Fetch ISS TLE Data ---
def fetch_iss_tle():
    url = "https://celestrak.org/NORAD/elements/stations.txt"
    response = requests.get(url)
    response.raise_for_status()
    lines = response.text.strip().splitlines()
    for i, line in enumerate(lines):
        if "ISS (ZARYA)" in line:
            return lines[i].strip(), lines[i+1].strip(), lines[i+2].strip()
    raise RuntimeError("ISS TLE not found.")
```

```
# --- STEP 2: Compute Subpoint Over Time ---
def compute_iss_positions(satellite, ts):
    now = datetime.utcnow().replace(tzinfo=utc)
    # Convert generator to a list of datetimes
    datetimes = [now + timedelta(minutes=i) for i in range(91)]
    times = ts.utc(datetimes)  # 90-minute range

    latitudes = []
    longitudes = []

    for t in times:
        subpoint = satellite.at(t).subpoint()
        latitudes.append(subpoint.latitude.degrees)
        longitudes.append(subpoint.longitude.degrees)

    return times, latitudes, longitudes

# --- STEP 3: Plotting on Cartopy World Map ---
def plot_ground_track(times, lats, lons, satellite, ts):
    fig = plt.figure(figsize=(12, 6))
    ax = plt.axes(projection=ccrs.PlateCarree())
    ax.stock_img()
    ax.add_feature(cfeature.BORDERS, linestyle=':')
    ax.coastlines()
    ax.gridlines(draw_labels=True, linestyle='--')

    # Plot ISS track
    ax.plot(lons, lats, color='dodgerblue', label='ISS Ground Track', linewidth=2)

    # Mark current position
    current_time = times[0]
    current_subpoint = satellite.at(current_time).subpoint()
    ax.scatter(current_subpoint.longitude.degrees,
               current_subpoint.latitude.degrees,
               color='red', s=50, label='Current Position', zorder=5)

    # Title and legend
    title_time = current_time.utc_datetime().strftime('%Y-%m-%d %H:%M:%S UTC')
    ax.set_title(f'ISS Ground Track (Next 90 Minutes)\nStart Time: {title_time}', fontsize=14)
    ax.legend(loc='lower left')

    plt.tight_layout()
    plt.show()

# --- MAIN PROGRAM ---
def main():
    print("🛰️  Plotting ISS Ground Track for the Next 90 Minutes...")

    # Load TLE
    name, line1, line2 = fetch_iss_tle()
    ts = load.timescale()
    satellite = EarthSatellite(line1, line2, name, ts)

    # Compute positions
    times, lats, lons = compute_iss_positions(satellite, ts)

    # Plot
    plot_ground_track(times, lats, lons, satellite, ts)

if __name__ == "__main__":
    main()
```

```
🛰️ Plotting ISS Ground Track for the Next 90 Minutes...
/usr/local/lib/python3.11/dist-packages/cartopy/io/__init__.py:241: DownloadWarning: Downloading: https://naturalearth.s3.amazonaws
  warnings.warn(f'Downloading: {url}', DownloadWarning)
/usr/local/lib/python3.11/dist-packages/cartopy/io/__init__.py:241: DownloadWarning: Downloading: https://naturalearth.s3.amazonaws
  warnings.warn(f'Downloading: {url}', DownloadWarning)
```



Task 2

```
!pip install skyfield
import csv
from datetime import datetime, timedelta
import pytz
import requests
from skyfield.api import load, EarthSatellite, wgs84, utc

# --- Load ISS TLE ---
def fetch_iss_tle():
    url = "https://celestrak.org/NORAD/elements/stations.txt"
    try:
        response = requests.get(url)
        response.raise_for_status()
        lines = response.text.strip().splitlines()
        for i, line in enumerate(lines):
            if "ISS (ZARYA)" in line:
                return lines[i], lines[i + 1], lines[i + 2]
    except Exception as e:
        print("Failed to fetch TLE data:", e)
        # Returning None or raising a more specific error might be better
        # depending on how the caller handles it. For now, exit as in original code.
        exit()
    raise RuntimeError("ISS TLE not found.")


# --- Find next full pass for a given location ---
def find_next_pass(lat, lon, satellite, ts):
    location = wgs84.latlon(lat, lon)
    t0 = ts.now()
    t1 = ts.utc((datetime.utcnow() + timedelta(days=1)).replace(tzinfo=utc))

    times, events = satellite.find_events(location, t0, t1, altitude_degrees=10.0)
    times = list(times)
    events = list(events)

    current_pass = {}
    for t, e in zip(times, events):
        if e == 0:
            current_pass = {"rise": t}
        elif e == 1 and "rise" in current_pass:
            current_pass["max"] = t
        elif e == 2 and "max" in current_pass:
            current_pass["set"] = t
            return current_pass  # Return only the first full pass
    return None

# --- Load city list from CSV ---
def load_locations(file_path):
    locations = []
    with open(file_path, newline='', encoding='utf-8-sig') as csvfile:
        reader = csv.DictReader(csvfile)
        # Print the fieldnames (column headers) to help diagnose the KeyError
        print("CSV Fieldnames:", reader.fieldnames)
        for row in reader:
            # Assuming the correct headers are "Location", "Latitude", "Longitude"
            # If the KeyError persists, check the output of reader.fieldnames
            # and update the keys below to match the actual headers in your CSV file.
            locations.append({
                "name": row["Location"],
                "lat": float(row["Latitude"]),
                "lon": float(row["Longitude"])
            })
    return locations

# --- Convert UTC to local time ---
def to_local(utc_time, timezone='Asia/Kolkata'):
    return utc_time.utc_datetime().replace(tzinfo=pytz.utc).astimezone(pytz.timezone(timezone))

# --- Main program ---
def main():
    ts = load.timescale()
    try:
        name, line1, line2 = fetch_iss_tle()
        satellite = EarthSatellite(line1, line2, name, ts)
    except Exception as e:
        print(f"Error fetching TLE data: {e}")
        return


    try:
        locations = load_locations("/content/locations.csv")
    except FileNotFoundError:
        print("Error: locations.csv not found. Please upload the file to /content/.")
        return
    except KeyError as e:
        print(f"Error loading locations: Missing expected column {e}. Please check the CSV headers.")
        return
    except ValueError as e:
        print(f"Error loading locations: Invalid data format in CSV - {e}. Ensure latitude and longitude are numbers.")
        return


    results = []
    print("\n🌍 ISS Next Pass Predictions:\n")
    print("{:<15} {:<20} {:<20} {:<20} {:<10}".format("Location", "Rise Time", "Culmination", "Set Time", "Duration"))

    for loc in locations:
        pass_data = find_next_pass(loc["lat"], loc["lon"], satellite, ts)
        if pass_data:
            rise = to_local(pass_data["rise"])
            max_ = to_local(pass_data["max"])
            set_ = to_local(pass_data["set"])
            duration = (set_ - rise).seconds
            results.append({
                "Location": loc["name"],
                "Rise Time": rise.strftime('%Y-%m-%d %H:%M:%S'),
                "Culmination": max_.strftime('%Y-%m-%d %H:%M:%S'),
                "Set Time": set_.strftime('%Y-%m-%d %H:%M:%S'),
                "Duration": duration
            })
            print(f"{loc['name']:<15} {rise:%Y-%m-%d %H:%M:%S}   {max_:%Y-%m-%d %H:%M:%S}   {set_:%Y-%m-%d %H:%M:%S}   {duration:<10}")
        else:
            print(f"{loc['name']:<15} No visible pass found in 24 hrs.")

    # Save to CSV
    try:
        with open("iss_pass_predictions.csv", "w", newline="") as csvfile:
            fieldnames = ["Location", "Rise Time", "Culmination", "Set Time", "Duration"]
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in results:
                writer.writerow(row)
        print("\n✅ Results saved to 'iss_pass_predictions.csv'.")
        print("\n✅ Results saved to 'iss_pass_predictions.csv'.")
    except IOError as e:
        print(f"Error saving results to CSV: {e}")


if __name__ == "__main__":
    main()
```

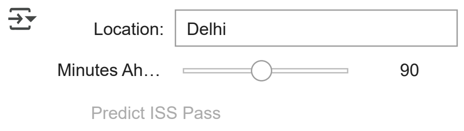
```
Requirement already satisfied: skyfield in /usr/local/lib/python3.11/dist-packages (1.53)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from skyfield) (2025.7.14)
Requirement already satisfied: jplephem>=2.13 in /usr/local/lib/python3.11/dist-packages (from skyfield) (2.23)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from skyfield) (2.0.2)
Requirement already satisfied: sgp4>=2.13 in /usr/local/lib/python3.11/dist-packages (from skyfield) (2.24)
CSV Fieldnames: ['Location', 'Latitude', 'Longitude']

🌍 ISS Next Pass Predictions:

Location        Rise Time            Culmination          Set Time             Duration
Delhi           2025-07-24 13:29:23   2025-07-24 13:32:23   2025-07-24 13:35:24   361
Mumbai          2025-07-24 13:26:28   2025-07-24 13:29:43   2025-07-24 13:32:59   390
Lucknow         2025-07-24 13:29:17   2025-07-24 13:32:35   2025-07-24 13:35:54   396
Bengaluru       2025-07-24 13:26:05   2025-07-24 13:29:11   2025-07-24 13:32:16   371
Kolkata         2025-07-24 11:54:49   2025-07-24 11:56:28   2025-07-24 11:58:06   197

✅ Results saved to 'iss_pass_predictions.csv'.
```

```
!pip install ipywidgets
```

```
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.11/dist-packages (7.7.1)
Requirement already satisfied: ipykernel>=4.5.1 in /usr/local/lib/python3.11/dist-packages (from ipywidgets) (6.17.1)
Requirement already satisfied: ipython-genutils~=0.2.0 in /usr/local/lib/python3.11/dist-packages (from ipywidgets) (0.2.0)
Requirement already satisfied: traitlets>=4.3.1 in /usr/local/lib/python3.11/dist-packages (from ipywidgets) (5.7.1)
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.11/dist-packages (from ipywidgets) (3.6.10)
Requirement already satisfied: ipython>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from ipywidgets) (7.34.0)
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from ipywidgets) (3.0.15)
Requirement already satisfied: debugpy>=1.0 in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1->ipywidgets) (1.8.15
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1->ipywidget
Requirement already satisfied: matplotlib-inline>=0.1 in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1->ipywidget
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1->ipywidgets) (1.6.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1->ipywidgets) (25.0)
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1->ipywidgets) (5.9.5)
Requirement already satisfied: pyzmq>=17 in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1->ipywidgets) (24.0.1)
Requirement already satisfied: tornado>=6.1 in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1->ipywidgets) (6.4.2)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.11/dist-packages (from ipython>=4.0.0->ipywidgets) (75.2
Collecting jedi>=0.16 (from ipython>=4.0.0->ipywidgets)
  Downloading jedi-0.19.2-py2.py3-none-any.whl.metadata (22 kB)
Requirement already satisfied: decorator in /usr/local/lib/python3.11/dist-packages (from ipython>=4.0.0->ipywidgets) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.11/dist-packages (from ipython>=4.0.0->ipywidgets) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from ipyth
Requirement already satisfied: pygments in /usr/local/lib/python3.11/dist-packages (from ipython>=4.0.0->ipywidgets) (2.19.2)
Requirement already satisfied: backcall in /usr/local/lib/python3.11/dist-packages (from ipython>=4.0.0->ipywidgets) (0.2.0)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.11/dist-packages (from ipython>=4.0.0->ipywidgets) (4.9.0)
Requirement already satisfied: notebook>=4.4.1 in /usr/local/lib/python3.11/dist-packages (from widgetsnbextension~=3.6.0->ipywidg
Requirement already satisfied: parso<0.9.0,>=0.8.4 in /usr/local/lib/python3.11/dist-packages (from jedi>=0.16->ipython>=4.0.0->ip
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.11/dist-packages (from jupyter-client>=6.1.12->ipyker
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.11/dist-packages (from jupyter-client>=6.1.12->ipyke
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0-
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.11/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3
Requirement already satisfied: nbformat in /usr/local/lib/python3.11/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.
Requirement already satisfied: nbconvert>=5 in /usr/local/lib/python3.11/dist-packages (from notebook>=4.4.1->widgetsnbextension~=
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.11/dist-packages (from notebook>=4.4.1->widgetsnbextens
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.11/dist-packages (from notebook>=4.4.1->widgetsnbextensi
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.11/dist-packages (from notebook>=4.4.1->widgetsnbextens
Requirement already satisfied: nbclassic>=0.4.7 in /usr/local/lib/python3.11/dist-packages (from notebook>=4.4.1->widgetsnbextensi
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.11/dist-packages (from pexpect>4.3->ipython>=4.0.0->ipywi
Requirement already satisfied: wcwidth in /usr/local/lib/python3.11/dist-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.11/dist-packages (from jupyter-core>=4.6.0->jupyter-cli
Requirement already satisfied: notebook-shim>=0.2.3 in /usr/local/lib/python3.11/dist-packages (from nbclassic>=0.4.7->notebook>=4
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook>=4.4.1->widg
Requirement already satisfied: bleach!=5.0.0 in /usr/local/lib/python3.11/dist-packages (from bleach[css]!=5.0.0->nbconvert>=5->no
Requirement already satisfied: defusedxml in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsn
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook>=4.4.1-
Requirement already satisfied: markupsafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook>=4.4.1->wid
Requirement already satisfied: mistune<4,>=2.0.3 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook>=4.4.1->w
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook>=4.4.1->wid
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook>=4.4.1
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.11/dist-packages (from nbformat->notebook>=4.4.1->wi
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.11/dist-packages (from nbformat->notebook>=4.4.1->widgets
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.1->jupyter-client>=6.1
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.11/dist-packages (from argon2-cffi->notebook>=4.4.1-
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from bleach!=5.0.0->bleach[css]!=5.0.0->nb
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from bleach[css]!=5.0.0->nbconvert
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat->notebook>
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat->not
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat->notebook
```

Task 3

```
# --- Imports ---
import pandas as pd
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display, clear_output
from datetime import datetime, timedelta
import pytz
import requests
from skyfield.api import load, EarthSatellite, wgs84, utc

# --- Load Locations from CSV ---
def load_locations(path="locations.csv"):
    return pd.read_csv(path, encoding='utf-8-sig')

locations_df = load_locations()
location_names = locations_df['Location'].tolist()

# --- Fetch TLE ---
def fetch_iss_tle():
    url = "https://celestrak.org/NORAD/elements/stations.txt"
    response = requests.get(url)
    lines = response.text.strip().splitlines()
    for i, line in enumerate(lines):
        if "ISS (ZARYA)" in line:
            return lines[i], lines[i + 1], lines[i + 2]
    raise RuntimeError("ISS TLE not found.")

# --- Predict next ISS pass ---
def find_next_pass(lat, lon, duration_minutes, satellite, ts):
    location = wgs84.latlon(lat, lon)

    for window in [duration_minutes, duration_minutes + 60, duration_minutes + 120]:
        t0 = ts.now()
        t1 = ts.utc((datetime.utcnow() + timedelta(minutes=window)).replace(tzinfo=utc))
        times, events = satellite.find_events(location, t0, t1, altitude_degrees=10.0)
        times = list(times)
        events = list(events)

        current_pass = {}
        for t, e in zip(times, events):
            if e == 0:
                current_pass = {"rise": t}
            elif e == 1 and "rise" in current_pass:
                current_pass["max"] = t
            elif e == 2 and "max" in current_pass:
                current_pass["set"] = t
                return current_pass
    return None


# --- Local time formatting ---
def to_local(utc_time, timezone='Asia/Kolkata'):
    return utc_time.utc_datetime().replace(tzinfo=pytz.utc).astimezone(pytz.timezone(timezone))

# --- Plot ---
def plot_pass(location_name, lat, lon, duration, satellite, ts):
    clear_output(wait=True)

    result = find_next_pass(lat, lon, duration, satellite, ts)

    if not result:
        print(f"No visible ISS pass in next {duration} minutes for {location_name}.")
        return

    rise = to_local(result["rise"])
    max_ = to_local(result["max"])
    set_ = to_local(result["set"])
    duration_sec = (set_ - rise).seconds
    alt, _, _ = (satellite - wgs84.latlon(lat, lon)).at(result["max"]).altaz()

    print(f"\n📍 **{location_name}** — Next ISS Pass")
    print(f"🔼 Rise Time     : {rise.strftime('%Y-%m-%d %H:%M:%S')}")
    print(f"🌞 Peak Time     : {max_.strftime('%Y-%m-%d %H:%M:%S')}")
    print(f"🔽 Set Time      : {set_.strftime('%Y-%m-%d %H:%M:%S')}")
    print(f"⏱ Duration       : {duration_sec} sec")
    print(f"📏 Peak Altitude : {alt.degrees:.2f}°")

    # Optional: plot a static line showing timing
    times = [rise, max_, set_]
    times = [rise, max_, set_]
    labels = ["Rise", "Max", "Set"]
    y = [10, alt.degrees, 10]

    fig, ax = plt.subplots(figsize=(6, 4))
    ax.plot(times, y, marker='o')
    ax.set_title(f"ISS Altitude Curve – {location_name}")
    ax.set_ylabel("Altitude (°)")
    ax.grid(True)
    for i, txt in enumerate(labels):
        ax.annotate(txt, (times[i], y[i]))
    plt.show()

# --- Interactive Widgets ---
location_dropdown = widgets.Dropdown(
    options=location_names,
    description="Location:"
)

duration_slider = widgets.IntSlider(
    value=90,
    min=10,
    max=180,
    step=10,
    description="Minutes Ahead:"
)

submit_button = widgets.Button(
    description="Predict ISS Pass",
    button_style='success'
)

# --- Load TLE and Timescale once ---
ts = load.timescale()
tle_name, tle1, tle2 = fetch_iss_tle()
satellite = EarthSatellite(tle1, tle2, tle_name, ts)

# --- Callback ---
def on_button_click(b):
    loc_name = location_dropdown.value
    row = locations_df[locations_df['Location'] == loc_name].iloc[0]
    lat, lon = row['Latitude'], row['Longitude']
    duration = duration_slider.value
    plot_pass(loc_name, lat, lon, duration, satellite, ts)

submit_button.on_click(on_button_click)

# --- Display Interface ---
display(widgets.VBox([
    location_dropdown,
    duration_slider,
    submit_button
]))
```

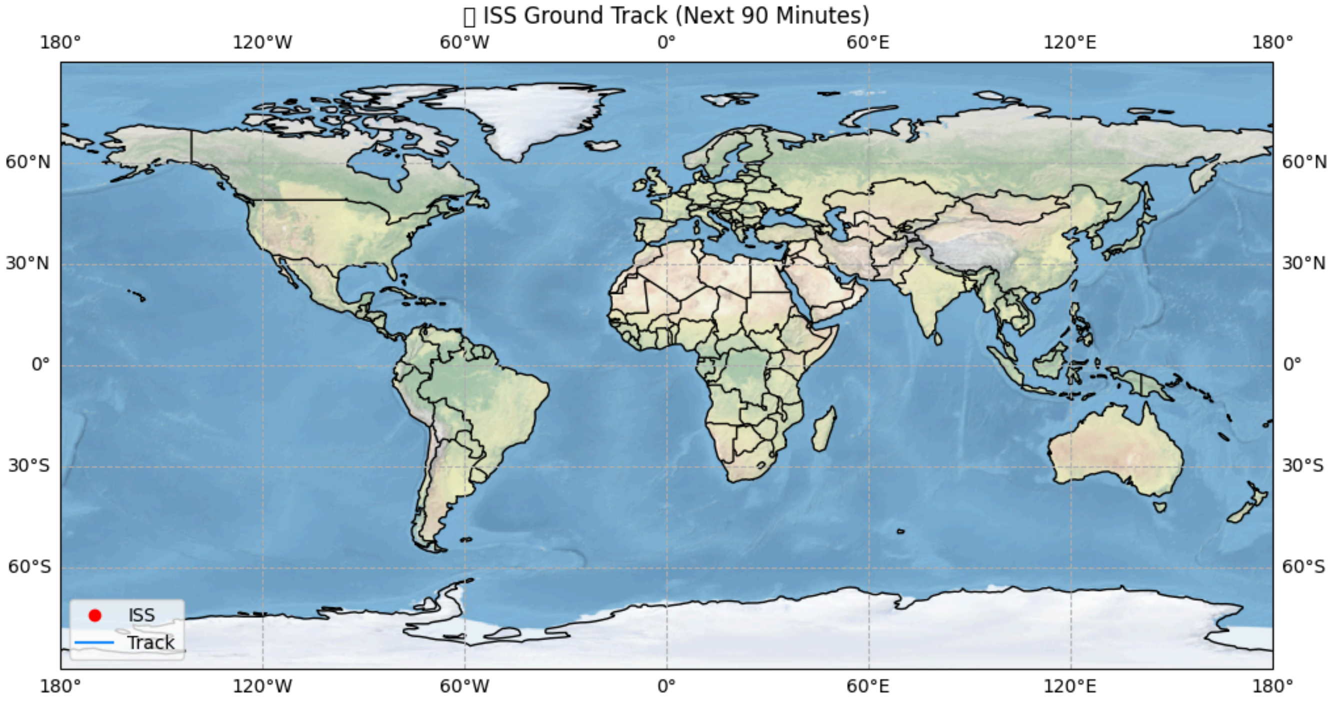


Task 4

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from datetime import datetime, timedelta
import requests
from skyfield.api import load, EarthSatellite, wgs84, utc

# --- Fetch ISS TLE ---
def fetch_iss_tle():
    url = "https://celestrak.org/NORAD/elements/stations.txt"
    response = requests.get(url)
    lines = response.text.strip().splitlines()
    for i, line in enumerate(lines):
        if "ISS (ZARYA)" in line:
            return lines[i], lines[i+1], lines[i+2]
    raise RuntimeError("ISS TLE not found.")
```

```
# --- Compute ISS subpoints ---
def get_iss_positions():
    ts = load.timescale()
    tle_name, tle1, tle2 = fetch_iss_tle()
    satellite = EarthSatellite(tle1, tle2, tle_name, ts)

    now = datetime.utcnow().replace(tzinfo=utc)
    # Create a list of datetime objects from the generator
    datetimes = [now + timedelta(minutes=i) for i in range(91)]
    times = ts.utc(datetimes)

    lats, lons = [], []
    for t in times:
        subpoint = satellite.at(t).subpoint()
        lats.append(subpoint.latitude.degrees)
        lons.append(subpoint.longitude.degrees)

    # Normalize longitudes
    lons = [(lon + 180) % 360 - 180 for lon in lons]
    return lats, lons

# --- Animate ---
def animate_iss(latitudes, longitudes):
    fig = plt.figure(figsize=(12, 6))
    ax = plt.axes(projection=ccrs.PlateCarree())
    ax.set_global()
    ax.stock_img()
    ax.add_feature(cfeature.BORDERS)
    ax.coastlines()
    ax.gridlines(draw_labels=True, linestyle="--")

    iss_dot, = ax.plot([], [], 'ro', markersize=6, transform=ccrs.PlateCarree(), label='ISS')
    path_line, = ax.plot([], [], 'dodgerblue', linewidth=1.5, transform=ccrs.PlateCarree(), label='Track')

    plt.legend(loc='lower left')
    plt.title('🌍 ISS Ground Track (Next 90 Minutes)')

    def init():
        iss_dot.set_data([], [])
        path_line.set_data([], [])
        return iss_dot, path_line

    def update(frame):
        iss_dot.set_data(longitudes[frame], latitudes[frame])
        path_line.set_data(longitudes[:frame+1], latitudes[:frame+1])
        return iss_dot, path_line

    ani = FuncAnimation(fig, update, frames=len(latitudes),
                        init_func=init, interval=200, blit=True, repeat=False)

    plt.show()

# --- Run ---
def main():
    print("🛰️ Animating ISS Path...")
    lats, lons = get_iss_positions()
    animate_iss(lats, lons)

main()
```

```
🛰️ Animating ISS Path...
/usr/local/lib/python3.11/dist-packages/cartopy/mpl/geoaxes.py:527: UserWarning: Glyph 127757 (\N{EARTH GLOBE EUROPE-AFRICA}) missin
  super()._update_title_position(renderer)
/usr/local/lib/python3.11/dist-packages/cartopy/mpl/geoaxes.py:524: UserWarning: Glyph 127757 (\N{EARTH GLOBE EUROPE-AFRICA}) missin
  return super().draw(renderer=renderer, **kwargs)
/usr/local/lib/python3.11/dist-packages/cartopy/mpl/geoaxes.py:527: UserWarning: Glyph 127757 (\N{EARTH GLOBE EUROPE-AFRICA}) missin
  super()._update_title_position(renderer)
/usr/local/lib/python3.11/dist-packages/cartopy/mpl/geoaxes.py:524: UserWarning: Glyph 127757 (\N{EARTH GLOBE EUROPE-AFRICA}) missin
  return super().draw(renderer=renderer, **kwargs)
```



```
!pip install astroquery
```

```
Collecting astroquery
  Downloading astroquery-0.4.10-py3-none-any.whl.metadata (6.3 kB)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.11/dist-packages (from astroquery) (2.0.2)
Requirement already satisfied: astropy>=5.0 in /usr/local/lib/python3.11/dist-packages (from astroquery) (7.1.0)
Requirement already satisfied: requests>=2.19 in /usr/local/lib/python3.11/dist-packages (from astroquery) (2.32.3)
Requirement already satisfied: beautifulsoup4>=4.8 in /usr/local/lib/python3.11/dist-packages (from astroquery) (4.13.4)
Requirement already satisfied: html5lib>=0.999 in /usr/local/lib/python3.11/dist-packages (from astroquery) (1.1)
Requirement already satisfied: keyring>=15.0 in /usr/local/lib/python3.11/dist-packages (from astroquery) (25.6.0)
Collecting pyvo>=1.5 (from astroquery)
  Downloading pyvo-1.7-py3-none-any.whl.metadata (4.7 kB)
Requirement already satisfied: pyerfa>=2.0.1.1 in /usr/local/lib/python3.11/dist-packages (from astropy>=5.0->astroquery) (2.0.1.5)
Requirement already satisfied: astropy-iers-data>=0.2025.4.28.0.37.27 in /usr/local/lib/python3.11/dist-packages (from astropy>=5.0
Requirement already satisfied: PyYAML>=6.0.0 in /usr/local/lib/python3.11/dist-packages (from astropy>=5.0->astroquery) (6.0.2)
Requirement already satisfied: packaging>=22.0.0 in /usr/local/lib/python3.11/dist-packages (from astropy>=5.0->astroquery) (25.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.8->astroquery) (2.7
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.8->astro
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.11/dist-packages (from html5lib>=0.999->astroquery) (1.17.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from html5lib>=0.999->astroquery) (0.5.1)
Requirement already satisfied: SecretStorage>=3.2 in /usr/local/lib/python3.11/dist-packages (from keyring>=15.0->astroquery) (3.3.
Requirement already satisfied: jeepney>=0.4.2 in /usr/local/lib/python3.11/dist-packages (from keyring>=15.0->astroquery) (0.9.0)
Requirement already satisfied: importlib_metadata>=4.11.4 in /usr/local/lib/python3.11/dist-packages (from keyring>=15.0->astroquer
Requirement already satisfied: jaraco.classes in /usr/local/lib/python3.11/dist-packages (from keyring>=15.0->astroquery) (3.4.0)
Requirement already satisfied: jaraco.functools in /usr/local/lib/python3.11/dist-packages (from keyring>=15.0->astroquery) (4.2.1)
Requirement already satisfied: jaraco.context in /usr/local/lib/python3.11/dist-packages (from keyring>=15.0->astroquery) (6.0.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19->astroquery
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19->astroquery) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19->astroquery) (2.5
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19->astroquery) (2025
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.11/dist-packages (from importlib_metadata>=4.11.4->keyring>=15.
Requirement already satisfied: cryptography>=2.0 in /usr/local/lib/python3.11/dist-packages (from SecretStorage>=3.2->keyring>=15.0
Requirement already satisfied: more-itertools in /usr/local/lib/python3.11/dist-packages (from jaraco.classes->keyring>=15.0->astro
Requirement already satisfied: backports.tarfile in /usr/local/lib/python3.11/dist-packages (from jaraco.context->keyring>=15.0->as
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.11/dist-packages (from cryptography>=2.0->SecretStorage>=3.2->k
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.12->cryptography>=2.0->SecretStor
Downloading astroquery-0.4.10-py3-none-any.whl (11.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.1/11.1 MB 70.9 MB/s eta 0:00:00
Downloading pyvo-1.7-py3-none-any.whl (1.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 45.5 MB/s eta 0:00:00
Installing collected packages: pyvo, astroquery
Successfully installed astroquery-0.4.10 pyvo-1.7
```

Task 5

```
import requests

url = "https://celestrak.org/NORAD/elements/stations.txt"
response = requests.get(url)

if response.status_code == 200:
    tle_lines = response.text.strip().split('\n')
    print("Satellite Name:", tle_lines[0])
    print("TLE Line 1:", tle_lines[1])
    print("TLE Line 2:", tle_lines[2])
else:
    print("Error downloading TLE:", response.status_code)
```

```
Satellite Name: ISS (ZARYA)
TLE Line 1: 1 25544U 98067A   25204.25047604  .00008934  00000+0  16419-3 0  9998
TLE Line 2: 2 25544  51.6351 128.3174 0002345 109.2368 347.8357 15.50024026520753
```

Task 6

```
from skyfield.api import EarthSatellite, load

# Assume you already have tle_lines[1] and tle_lines[2]
satellite = EarthSatellite(tle_lines[1], tle_lines[2], tle_lines[0])
ts = load.timescale()
t = ts.now()
geocentric = satellite.at(t)
print("ISS position (km):", geocentric.position.km)
```

```
ISS position (km): [ 2504.14646549 -6040.78475421  1836.27761373]
```

Task 7

```
import requests
from datetime import datetime
from skyfield.api import load, EarthSatellite
import matplotlib.pyplot as plt

# Step 1: Fetch latest TLE data from Celestrak
url = "https://celestrak.org/NORAD/elements/stations.txt"
response = requests.get(url)

if response.status_code == 200:
    tle_lines = response.text.strip().split('\n')
    if len(tle_lines) >= 3:
        name = tle_lines[0].strip()
        line1 = tle_lines[1].strip()
        line2 = tle_lines[2].strip()
    else:
        raise ValueError("Malformed TLE response")
else:
    raise ConnectionError(f"Error downloading TLE: {response.status_code}")

# Step 2: Parse using Skyfield
ts = load.timescale()
satellite = EarthSatellite(line1, line2, name, ts)
orb = satellite.model

# Step 3: Compute orbital elements
inclination_deg = orb.inclo * (180 / 3.1415926)   # Radians to degrees
eccentricity = orb.ecco
orbital_period_min = 1440.0 / orb.no_kozai        # Period = 1 / revs per day

# Step 4: Display results
print("\n🛰️ ISS Orbital Elements (from latest TLE):")
print(f"Satellite Name     : {name}")
print(f"Inclination (deg)  : {inclination_deg:.3f}")
print(f"Eccentricity       : {eccentricity:.7f}")
print(f"Orbital Period (min): {orbital_period_min:.2f}")

# Step 5: Optional single-point plot (text visualization)
plt.figure(figsize=(6, 4))
plt.bar(['Inclination (°)', 'Eccentricity', 'Period (min)'],
        [inclination_deg, eccentricity * 100, orbital_period_min],
        color=['skyblue', 'green', 'orange'])

plt.title(f'ISS Orbital Elements - {datetime.utcnow().date()}')
plt.ylabel('Value')
plt.grid(True, axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
🛰 ISS Orbital Elements (from latest TLE):
Satellite Name     : ISS (ZARYA)
```

ISS TRACKER APP CODE

```
import streamlit as st
from streamlit_folium import st_folium
from streamlit_autorefresh import st_autorefresh
import folium
import requests
from skyfield.api import Loader, EarthSatellite, wgs84
from datetime import datetime

# Set page config
st.set_page_config(page_title="ISS Tracker", layout="wide")

# Auto-refresh every 60 seconds
st_autorefresh(interval=60000, key="datarefresh")

# Skyfield loader
load = Loader('./skyfield_data')
ts = load.timescale()

# --- CACHED: Fetch TLE Data ---
@st.cache_data(ttl=60)
def fetch_iss_tle():
    url = "https://celestrak.org/NORAD/elements/stations.txt"
    response = requests.get(url)
    if response.status_code == 200:
        tle_lines = response.text.strip().split('\n')
        if len(tle_lines) >= 3:
            return tle_lines[0], tle_lines[1], tle_lines[2]
    return None, None, None

# --- Calculate Current ISS Location ---
def get_iss_position():
    name, line1, line2 = fetch_iss_tle()
    if not name:
        return None, None, "TLE fetch failed"

    satellite = EarthSatellite(line1, line2, name, ts)
    t = ts.now()
    subpoint = satellite.at(t).subpoint()  # ✅ FIXED HERE
    latitude = subpoint.latitude.degrees
    longitude = subpoint.longitude.degrees
    return latitude, longitude, f"{name} at {t.utc_datetime():%Y-%m-%d %H:%M:%S} UTC"

# --- Display ISS Position ---
st.title("🌍 Live ISS Tracker")

lat, lon, status = get_iss_position()

if lat is not None and lon is not None:
    st.markdown(f"**📍 ISS Current Position:** Latitude = `{lat:.2f}`, Longitude = `{lon:.2f}`")
    st.markdown(f"**📅 Status:** {status}")

    # Map setup
    m = folium.Map(location=[lat, lon], zoom_start=3)
    folium.Marker([lat, lon], tooltip="ISS", icon=folium.Icon(color="red", icon="rocket", prefix="fa")).add_to(m)
    st_folium(m, width=700, height=500)

else:
    st.error("Unable to fetch ISS location.")
```

Explanation:

```
import streamlit as st
from streamlit_folium import st_folium
from streamlit_autorefresh import st_autorefresh
import folium
import requests
from skyfield.api import Loader, EarthSatellite, wgs84
from datetime import datetime
```

streamlit: For building the web interface.

streamlit_folium: To display interactive maps using Folium in Streamlit.

streamlit_autorefresh: Auto-refreshes the Streamlit app at set intervals.

folium: For map visualization.

requests: To fetch TLE (satellite data) from the internet.

skyfield: Astronomy package used to compute satellite positions.

datetime: To format and manage timestamps.

```
st.set_page_config(page_title="ISS Tracker", layout="wide")
```

Sets the browser tab title to "ISS Tracker".

Uses a wide layout for a better visual appearance.

```
st_autorefresh(interval=60000, key="datarefresh")
```

Refreshes the app **every 60,000 milliseconds = 60 seconds**.

Useful to keep the ISS position updated in near real-time.

```
load = Loader('./skyfield_data')
ts = load.timescale()
```

Loader downloads necessary astronomical data and stores it in ./skyfield_data.

ts is the timescale object used to get current times for satellite position calculations.

```
@st.cache_data(ttl=60)
def fetch_iss_tle():
url = "https://celestrak.org/NORAD/elements/stations.txt"
```

Downloads satellite orbital data (TLE) from **Celestrak** (a reliable satellite tracking source).

@st.cache_data(ttl=60): Caches the result for 60 seconds to reduce unnecessary network calls.

Returns the TLE lines needed to define the ISS orbit.

```
def get_iss_position():
    subpoint = satellite.at(t).subpoint()
```

Creates an EarthSatellite object from the TLE.

Uses t = ts.now() to get the current UTC time.

Computes the **subpoint** — the point on Earth directly beneath the satellite.

Extracts:

- latitude in degrees
- longitude in degrees

Returns them along with a formatted status message.

```
st.title("🌍 Live ISS Tracker")
```

Adds a title to the app.

```
if lat is not None and lon is not None:
    st.markdown(...)
```

Displays:

- ISS latitude and longitude (rounded to 2 decimal places).
- UTC timestamp when this position was calculated.

```
m = folium.Map(location=[lat, lon], zoom_start=3)
folium.Marker(...).add_to(m)
st_folium(m, width=700, height=500)
```

Creates a folium map centered on ISS coordinates.

Adds a red rocket icon to indicate the ISS position.

Displays the map inside the Streamlit app using st_folium.

```
else:
    st.error("Unable to fetch ISS location.")
```

If the TLE data fetch fails or the coordinates aren't valid, it shows an error message.

✅ **Final Result:**

You get a **live map-based ISS tracker** that:

- Refreshes every 60 seconds.
- Shows the ISS’s real-time location on a world map.
- Displays current coordinates and timestamp.

🌍 Live ISS Tracker

📍 **ISS Current Position:** Latitude = `44.88` , Longitude = `-61.63`

📅 **Status:** ISS (ZARYA)
at 2025-07-23 19:47:41 UTC