Day 3: Comprehensive Guide and Notes

Focus Areas

1. **Network Programming**
   - Basics of sockets.
   - REST API concepts and usage.
2. **File Processing**
   - Reading and writing CSV, JSON, and XML files.
   - Integrating data from multiple formats.

1. Network Programming

1.1 Sockets

Sockets provide a way to connect two devices over a network. A socket can act as a client or a server.

Key Concepts:

- **Socket Address**: Combination of IP address and port number.
- **Socket Types**:
  - SOCK_STREAM: TCP socket (reliable, connection-oriented).
  - SOCK_DGRAM: UDP socket (unreliable, connectionless).

Example: Simple Echo Server and Client:

- The **server** listens for connections and echoes back any message received.
- The **client** connects to the server and sends messages.

1.2 REST API

REST APIs allow interaction with web services using HTTP methods:

- GET: Fetch data.
- POST: Submit data.
- PUT: Update data.
- DELETE: Remove data.

Example: Fetching Data Using Python’s requests Library:

python

```
import requests


url = "https://jsonplaceholder.typicode.com/posts"
response = requests.get(url)
if response.status_code == 200:
   data = response.json()
   print(data[:5])  # Display the first 5 posts
else:
   print(f"Error: {response.status_code}")
```

2. File Processing

2.1 CSV Files

CSV (Comma-Separated Values) files are plain text files used to store tabular data.

Operations:

- **Reading CSV**: Use csv.DictReader to read the file and convert rows into dictionaries.
- **Writing CSV**: Use csv.DictWriter to write data into a CSV file.

Example:

```
python

import csv


# Reading a CSV file

with open('data.csv', mode='r') as file:

    reader = csv.DictReader(file)

    for row in reader:

        print(row)


# Writing to a CSV file

with open('output.csv', mode='w', newline='') as file:

    writer = csv.DictWriter(file, fieldnames=['Name', 'Age', 'City'])

    writer.writeheader()

    writer.writerow({'Name': 'John', 'Age': 30, 'City': 'New York'})
```

2.2 JSON Files

JSON (JavaScript Object Notation) is a lightweight data-interchange format.

Operations:

- **Reading JSON**: Use json.load() to parse a file.
- **Writing JSON**: Use json.dump() to save data.

Example:

```
python

import json


# Reading JSON

with open('data.json', 'r') as file:

   data = json.load(file)

   print(data)


# Writing JSON

new_data = {"name": "John", "age": 30, "city": "New York"}

with open('output.json', 'w') as file:

   json.dump(new_data, file, indent=4)
```

2.3 XML Files

XML (eXtensible Markup Language) is used for structuring data.

Operations:

- **Parsing XML**: Use xml.etree.ElementTree to parse and navigate XML files.

Example:

python

```
import xml.etree.ElementTree as ET


tree = ET.parse('data.xml')
root = tree.getroot()
for child in root:
  print(child.tag, child.attrib)
```

---

3. Integration

Combining File Formats

Integrating data from CSV, JSON, and XML is common in applications. For example:

- Use CSV for production data.
- Use JSON for inventory levels.
- Use XML for supplier schedules.

Example:

1. Parse CSV to get production details.
2. Load JSON to update stock levels based on production output.
3. Parse XML to identify suppliers for restocking.

Potential Use of Third-Party Packages

1. Streamlit

Purpose: Build interactive web applications for data visualization and analysis using Python.

Example Use:

- Create dashboards to display production and inventory data in real-time.
- Visualize supplier schedules and maintenance logs interactively.

Basic Usage:

python

```
import streamlit as st


st.title("Production Monitoring Dashboard")

st.write("This is a simple Streamlit app.")
```

2. OpenCV

Purpose: Computer vision and image processing.

Example Use:

- Analyze images from production lines to detect defects.
- Count items on conveyor belts.

Basic Usage:

python

```
import cv2


image = cv2.imread('image.jpg')

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

cv2.imshow('Gray Image', gray)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

3. httpx

Purpose: Advanced HTTP client for Python with support for asynchronous requests.

Example Use:

- Fetch data from multiple APIs concurrently to update dashboards.
- Post updates to APIs faster using asynchronous calls.

Basic Usage:

python

```
import httpx


async def fetch_data(url):
    async with httpx.AsyncClient() as client:
        response = await client.get(url)
        return response.json()


# Usage with asyncio
import asyncio
asyncio.run(fetch_data("https://jsonplaceholder.typicode.com/posts"))
```

4. FastAPI

Purpose: Build high-performance APIs quickly and efficiently.

Example Use:

- Create an API endpoint for the production monitoring system.
- Expose supplier schedules and inventory levels via RESTful APIs.

Basic Usage:

```python
from fastapi import FastAPI
app = FastAPI()


@app.get("/")
def read_root():
  return {"message": "Welcome to the Production Monitoring API"}
```

Summary

1. **Network Programming**:
   - Master sockets for low-level communication.
   - Leverage REST APIs for integrating external services.
2. **File Processing**:
   - Gain proficiency in handling CSV, JSON, and XML files.
   - Integrate data from multiple sources for cohesive reporting.
3. **Third-Party Packages**:
   - **Streamlit**: Interactive dashboards.
   - **OpenCV**: Visual inspection and image analysis.
   - **httpx**: Efficient HTTP requests.
   - **FastAPI**: High-performance APIs for production systems.

This comprehensive guide ensures a solid foundation for Day 3 topics and extends learning with practical real-world applications using modern tools