Python Intensive Day 5: Mastering APIs, Data Analysis, Databases, and Final Capstone Projects

Python Intensive Day 5: Mastering APIs, Data Analysis, Databases, and Final Capstone Projects

---

Objective

By the end of **Day 5**, students will: 1. Master the **integration of APIs, files, and databases** with OOP. 2. Develop **real-world applications** using data analysis tools. 3. Complete **extensive hands-on projects** to reinforce learning. 4. Apply **OOP concepts holistically** by building a **final capstone project**.

---

1. Comprehensive API Integration with OOP (90 minutes)

Why Use APIs?

APIs provide real-time data access, such as weather reports, financial data, or stock prices. **OOP encapsulates API logic** into reusable classes, making applications more organized and scalable.

Step-by-Step API Client Implementation

```
import requests

class APIClient:
    def __init__(self, base_url):
        self.base_url = base_url

    def get(self, endpoint, params=None):
        response = requests.get(self.base_url + endpoint, params=params)
        if response.status_code == 200:
            return response.json()
        else:
            print(f"Error {response.status_code}: {response.text}")
            return None
```

Explanation: - **`requests.get()`**: Makes a GET request to the API. - **Encapsulating API calls**: Encapsulation ensures you don't duplicate API logic in multiple places.

Usage Example

```
client = APIClient("https://api.openweathermap.org/data/2.5/")
params = {"q": "London", "appid": "YOUR_API_KEY"}
data = client.get("weather", params)
print(data)
```

---

2. Project 1: Weather App Using OpenWeather API (45 minutes)

Goal:

Build a weather app that: 1. **Fetches real-time weather data**. 2. Displays **temperature, humidity, and conditions**. 3. Suggests **appropriate attire**.

```
class WeatherApp:
    def __init__(self, api_key):
        self.client = APIClient("https://api.openweathermap.org/data/2.5/")
        self.api_key = api_key

    def get_weather(self, city):
        params = {"q": city, "appid": self.api_key, "units": "metric"}
        data = self.client.get("weather", params)
        if data:
            temp = data['main']['temp']
            condition = data['weather'][0]['description']
            print(f"Weather in {city}: {temp}°C, {condition}")

# Usage
app = WeatherApp("YOUR_API_KEY")
app.get_weather("New York")
```

Challenge:
- Add a **forecast feature** for the next 5 days using OpenWeather's forecast endpoint.

---

3. Advanced File Handling Using OOP (75 minutes)

Why Use Files?

Files store data **persistently** across program runs. Using **OOP for file management** ensures organized and reusable code.

Building a FileHandler Class

```
class FileHandler:
    def __init__(self, filename):
        self.filename = filename

    def write(self, data):
        with open(self.filename, 'w') as file:
            file.write(data)

    def read(self):
        try:
            with open(self.filename, 'r') as file:
                return file.readlines()
        except FileNotFoundError:
            print("File not found.")
            return []
```

Project 2: Persistent To-Do List Manager (45 minutes)

Goal:
Build a **To-Do List Manager** with: 1. **Tasks stored in a file**. 2. **Add, view, and delete tasks**. 3. Automatically **load tasks on startup**.

```
class ToDoList:
    def __init__(self, filename):
        self.handler = FileHandler(filename)
        self.tasks = self.load_tasks()

    def load_tasks(self):
        return self.handler.read()

    def add_task(self, task):
        self.tasks.append(task)
        self.save_tasks()

    def save_tasks(self):
        self.handler.write('\n'.join(self.tasks))

    def display_tasks(self):
        print("Your To-Do List:")
        for task in self.tasks:
            print(f"- {task}")

# Example Usage
todo = ToDoList("tasks.txt")
todo.add_task("Complete Python project")
todo.display_tasks()
```

Challenge:
- Add a **feature to mark tasks as completed**.

---

4. Data Analysis Using Pandas with OOP (75 minutes)

Why Use Pandas with OOP?

Pandas simplifies data manipulation, while OOP ensures that the data pipeline is **modular** and reusable.

Building a DataCleaner Class

```
import pandas as pd

class DataCleaner:
    def __init__(self, filepath):
        self.df = pd.read_csv(filepath)

    def remove_duplicates(self):
        self.df = self.df.drop_duplicates()

    def fill_missing(self, value):
        self.df = self.df.fillna(value)

    def save_cleaned_data(self, output_path):
        self.df.to_csv(output_path, index=False)

    def display_data(self):
        print(self.df.head())
```

Project 3: Data Cleaning and Reporting Tool (60 minutes)

Goal:
Create a tool that: 1. Loads **CSV data**. 2. **Removes duplicates**. 3. **Fills missing values**. 4. **Exports cleaned data** to a new file.

---

5. Database Integration Using SQLite and OOP (75 minutes)

Why Use Databases?

Databases ensure **organized, persistent data storage**. SQLite is a lightweight, file-based database that integrates seamlessly with Python.

Building a Database Class

```
import sqlite3

class Database:
    def __init__(self, db_name):
        self.conn = sqlite3.connect(db_name)
        self.cursor = self.conn.cursor()

    def create_table(self, query):
        self.cursor.execute(query)
        self.conn.commit()

    def insert(self, query, values):
        self.cursor.execute(query, values)
        self.conn.commit()

    def fetch_all(self, query):
        self.cursor.execute(query)
        return self.cursor.fetchall()
```

Project 4: Library Management System with SQLite (75 minutes)

Goal:
Develop a **Library Management System** that: 1. **Stores books in a database**. 2. **Allows users to borrow and return books**. 3. Displays **available books**.

```
db = Database("library.db")
db.create_table("CREATE TABLE IF NOT EXISTS books (id INTEGER PRIMARY KEY,
title TEXT, author TEXT)")

db.insert("INSERT INTO books (title, author) VALUES (?, ?)", ("Python 101",
"Eric Matthes"))
books = db.fetch_all("SELECT * FROM books")
print(books)
```

---

6. Capstone Project: Stock Portfolio Tracker with APIs and Databases (90 minutes)

Goal:
Create a **Portfolio Tracker** that: 1. Fetches **real-time stock prices** using the **Alpha Vantage API**. 2. **Stores portfolio data** in SQLite. 3. **Calculates the total value** of the portfolio.

```
class PortfolioTracker:
    def __init__(self, api_key):
        self.client = APIClient("https://www.alphavantage.co/query?")
        self.api_key = api_key
        self.db = Database("portfolio.db")
        self.db.create_table("CREATE TABLE IF NOT EXISTS stocks (symbol TEXT,
shares INTEGER)")

    def add_stock(self, symbol, shares):
        self.db.insert("INSERT INTO stocks (symbol, shares) VALUES (?, ?)",
(symbol, shares))

    def get_portfolio_value(self):
        stocks = self.db.fetch_all("SELECT * FROM stocks")
        total_value = 0
        for symbol, shares in stocks:
            data = self.client.get("", {"function": "GLOBAL_QUOTE", "symbol":
symbol, "apikey": self.api_key})
            price = float(data["Global Quote"]["05. price"])
            total_value += price * shares
        print(f"Portfolio Value: ${total_value:.2f}")

# Usage
tracker = PortfolioTracker("YOUR_API_KEY")
tracker.add_stock("AAPL", 10)
tracker.get_portfolio_value()
```

---

7. Recap, Challenges, and Final Homework (15 minutes)

Recap:

- **Integrated APIs, files, databases**, and **data analysis** using OOP.
- Completed **5+ major projects**, including:
  - Weather App
  - To-Do List Manager
  - Library System
  - Portfolio Tracker

Final Homework:

1. **Extend the Portfolio Tracker** to handle **cryptocurrency**.
2. **Build a Student Information System** using SQLite and APIs.

**

Day 5 Summary**

- Completed **5+ detailed projects** and a **capstone project**.
- Integrated **OOP principles** with real-world tools (APIs, SQLite, Pandas).
- Students are now equipped to tackle **larger-scale projects** independently.