Generative AI

Trainer: Nilesh Ghule

Python Modules: Organizing Your Code

- What are modules?
  - Reusable files containing Python code (functions, classes, variables)
- Why use modules?
  - Organize code, avoid repetition, share functionality between projects
- Built-in modules: Python comes with many useful modules
  - math
  - random
  - datetime
  - os

Creating and Using Your Own Modules

- Creating a module: Save Python code in a .py file
- Importing modules: Use the 'import' keyword to access module contents
- Different import styles:
  - import module_name
  - from module_name import function_name
  - from module_name import *
    - (imports everything)

```
# Example: Creating and using a module
# File: calculator.py
def add(a, b):
    return a + b

def multiply(a, b):
    return a * b
```

```
# File: main.py
import calculator
result = calculator.add(5, 3)  # Returns 8

# Or using specific imports
from calculator import multiply
result = multiply(5, 3)  # Returns 15
```

Popular Python Modules for Everyday Use

- math: Mathematical functions (sqrt, sin, cos, pi)
- random: Generate random numbers and selections
- datetime: Work with dates and times
- os: Interact with operating system
- json: Work with JSON data

```
# Examples of built-in modules
import math
import random
import datetime
import os

# Using math module
print(math.sqrt(16))   # 4.0
print(math.pi)         # 3.141592653589793

# Using random module
print(random.randint(1, 10))  # Random number
between 1 and 10

# Using datetime module
now = datetime.datetime.now()
print(f"Current date and time: {now}")

# Using os module
print(f"Current directory: {os.getcwd()}")
```

Installing External Modules with pip

- pip: Python's package installer (comes with Python 3.4+)
- Finding packages: PyPI (Python Package Index) - thousands of free packages
- Basic pip commands:
  - pip install package_name
  - pip install package_name==version_number
  - pip list (show installed packages)
  - pip uninstall package_name

The Problem: Python Package Conflicts

- Different projects need different package versions
  - Project A - "requests" package version 2.20.0
  - Project B - "requests" package version 2.28.1
- Global package installation can cause conflicts
- Solution: Virtual Environments - isolated Python environments per project
- Each virtual environment has its own:
  - Python interpreter
  - Installed packages
  - site-packages directory → external installed pkgs

pip install

Virtual Environments on Windows

- Built-in module: venv (included with Python 3.3+)
- Creating a virtual environment:
  - python -m venv env_name
  - Common folder names: 'venv', 'env', '.venv'
- Activating the environment (Windows Command Prompt):
  - env_name\Scripts\activate
- Activating (Windows PowerShell):
  - env_name\Scripts\Activate.ps1

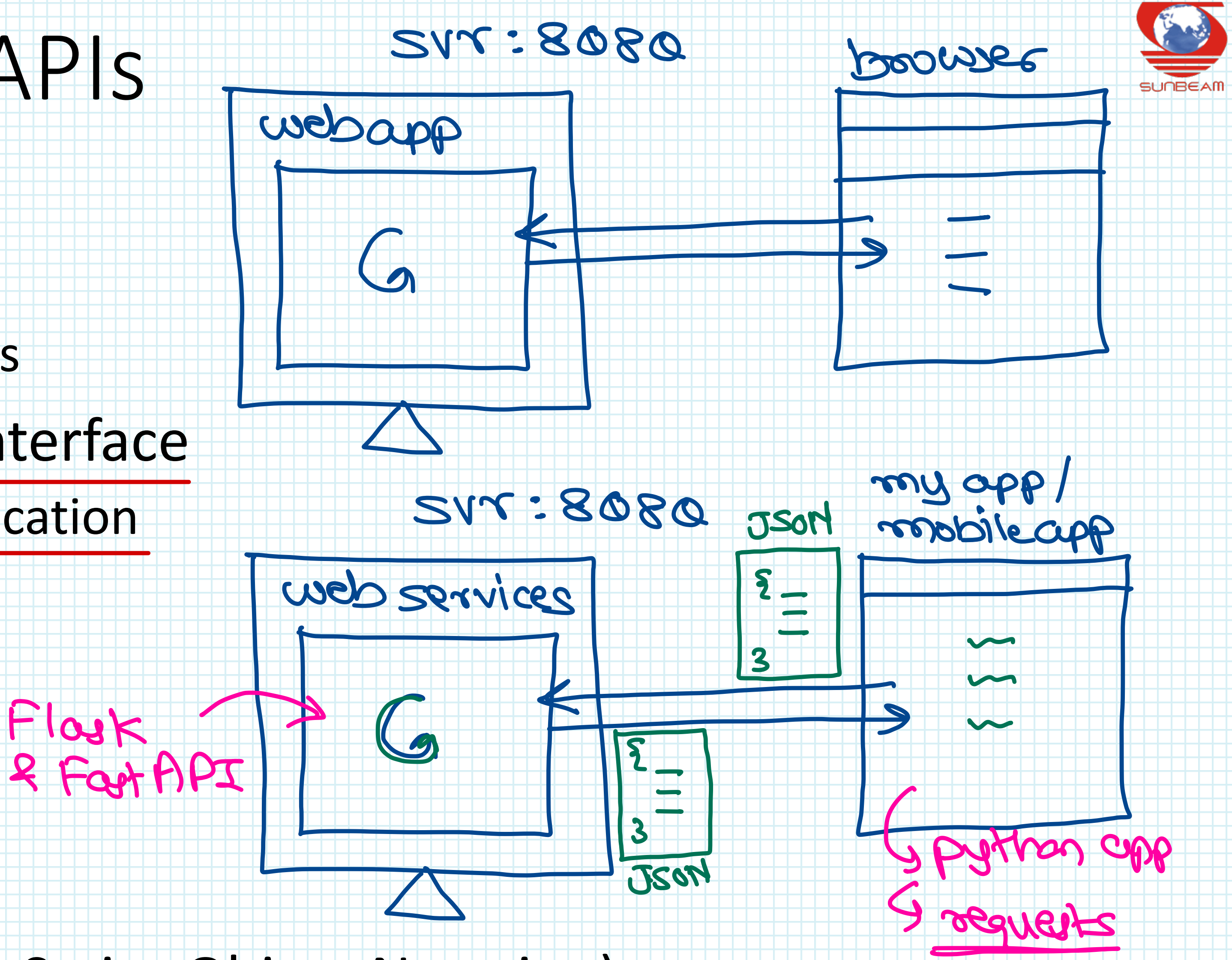
Working with Virtual Environments

- Check if activated: Command prompt shows (venv) prefix
- Installing packages in virtual environment:
    - pip install package_name (installs only in current venv)
- Deactivating the environment:
    - deactivate
- Saving requirements:
    - pip freeze > requirements.txt
- Installing from requirements:
    - pip install -r requirements.txt

```
# Complete workflow example
# Create virtual environment
python -m venv myproject_env

# Activate it (Windows CMD)
myproject_env\Scripts\activate

# Install packages
pip install requests pandas

# Save requirements
pip freeze > requirements.txt

# Deactivate when done
deactivate
```

Introduction to REST APIs

- What is REST?
  - **Representational State Transfer**
  - architectural style for web services
- API: Application Programming Interface
  - set of rules for software communication
- REST APIs use HTTP methods:
  - GET: Retrieve data
  - POST: Create new data
  - PUT: Update existing data
  - DELETE: Remove data
- Data format: Typically JSON (JavaScript Object Notation)



Installing and Importing the Requests Module

- requests: Popular Python module for making HTTP requests
- Not built-in - needs to be installed: cmd> pip install requests
- Importing in python app: import requests
- Basic structure of a request:
  - response = requests.method(url, params, headers, data, json)

```
Example:

# Basic GET request
response = requests.get('https://api.github.com')

# Check if request was successful
print(f"Status Code: {response.status_code}")  # 200 means success
print(f"Response: {response.text[:100]}...")  # First 100 characters
```

Making GET Requests

- GET: Retrieve data from a server
- Common uses: Fetching web pages, API data
- Adding parameters:
  - URL query parameters (?key=value)
- Response object properties:
  - status_code: HTTP status (200, 404, 500)
  - text: Response content as text
  - json(): Parse JSON response to Python dictionary
  - headers: Response headers

```
import requests

# GET request with parameters
url = "https://jsonplaceholder.typicode.com/posts"

# Add parameters to URL
response1 = requests.get(url + "?userId=1")

# Working with the response
print(f"Status: {response1.status_code}")
print(f"Headers: {response1.headers['content-type']}")

# Parse JSON response
data = response1.json()
print(f"Number of posts: {len(data)}")
print(f"First post title: {data[0]['title']}")
```

Making POST Requests

- POST: Send data to create new resources
- Common uses: Submitting forms, creating records
- Sending data:
  - Usually JSON data: json parameter (automatically sets Content-Type)
- Adding headers:
  - Customize request with headers dictionary

```
import requests
import json

url = "https://jsonplaceholder.typicode.com/posts"

# Data to send
new_post = {
    "title": "My New Post",
    "body": "This is the content of my new post",
    "userId": 1
}

# Send with custom headers
headers = {
    "Content-Type": "application/json",
    "Authorization": "Bearer your_security_token"
}
response = requests.post(url, data=json.dumps(new_post),
headers=headers)
print(f"\nCustom headers status: {response.status_code}")
print(response.json())
```

Nilesh Ghule

Practical Example: Weather API

- Real-world example: Fetch weather data from OpenWeatherMap
  1. Get API key (free tier available)
  2. Construct API URL with parameters
  3. Make GET request
  4. Parse and display results

```
def get_current_weather(city_name):
    api_key="OPENAPI_KEY"
    base_url = "https://api.openweathermap.org/data/2.5/weather"
    complete_url = f"{base_url}?q={city_name}&appid={api_key}&units=metric"
    try:
        response = requests.get(complete_url)
        response.raise_for_status()  # Exception for bad status codes
        data = response.json()
        return data
    except requests.exceptions.RequestException as e:
        print(f"Error fetching weather data: {e}")
    except json.JSONDecodeError:
        print("Error decoding JSON response from the API.")
    return None
```

Complete Project Setup Example

- Putting it all together:
  - Create virtual environment
  - Install required packages
  - Organize code into modules
  - Make API calls
  - Save requirements for reproducibility

```
# 1. Create project folder
mkdir weather_app
cd weather_app

# 2. Create virtual environment
python -m venv venv

# 3. Activate virtual environment
# CMD:
venv\Scripts\activate

# 4. Install required packages
pip install requests python-dotenv
```

```
# 5. Create project structure
#   weather_app/
#    ├── venv/              # Virtual environment
#    ├── weather.py         # Main module
#    ├── utils.py           # Helper functions
#    ├── .env               # API keys (git ignore)
#    └── requirements.txt

# 6. Save requirements
pip freeze > requirements.txt

# 7. To share your project:
# - Share code files
# - Share requirements.txt
# - Others can: pip install -r requirements.txt
```

Summary and Key Takeaways

- Modules: Organize code into reusable files using import
- Virtual Environments: Isolate project dependencies using venv
- REST API Calls: Use requests module for GET, POST, PUT, DELETE
- Always: Handle errors, check status codes, use timeouts
- Best practice: Store API keys in .env files, not in code
- Generative AI creates new content rather than just classifying existing data.

Practice Exercises

- Exercise 1: Create a math_utils module with functions for area calculations
- Exercise 2: Set up a virtual environment and install requests and pandas
- Exercise 3: Fetch data from JSONPlaceholder API and save to a file
- Exercise 4: Create a weather app that takes city input and displays forecast
- Challenge: Build a complete app with modules, venv, and API integration

Thank You!

Nilesh Ghule

<nilesh@sunbeaminfo.com>