GitHub Copilot with Python & Flask for Web Application Development

Duration: 8 Hours

Context

GitHub Copilot accelerates backend and full-stack development by generating boilerplate code, suggesting API endpoints, writing tests, and helping with debugging. For freshers learning **Python with Flask**, Copilot can scaffold routes, generate HTML templates, integrate APIs, and suggest error handling patterns.

This program provides a hands-on journey to build a Flask-based web application that consumes the **JokeAPI** and displays jokes dynamically, while learning to use GitHub Copilot effectively and responsibly.

Key Outcomes

By the end of this program, participants will be able to:

- Install and configure GitHub Copilot in VS Code for Python development.
- Use Copilot for generating Python functions, Flask routes, and templates.
- Build a Flask web application structure (routes, templates, static files).
- Integrate an external REST API (JokeAPI) into the Flask app.
- Apply Copilot to generate error handling, logging, and input validation.
- Write unit tests with pytest using Copilot assistance.
- Apply Copilot to generate inline documentation and code refactoring.
- Deliver and demo a **functional Flask JokeApp** as a mini-project.

Pre-requisites

- Basic Python programming (functions, loops, imports).
- Familiarity with HTTP basics (requests, responses, JSON).

Indicative Content Coverage (Detailed)

Module	Topic Coverage	Duration
Getting Started with GitHub Copilot & Flask	• **Introduction to GitHub Copilot** - How Copilot assists Python development (function generation, error handling, debugging). • **Setup environment** - Install Python, pip, virtualenv, Flask; configure VS Code with Python extension and Copilot. • **Copilot basics** - Inline suggestions, multi-line completions, prompting with comments. • **First Flask app** - Use Copilot to scaffold “Hello World” app (app.py with one route), run with flask run, auto-reload with debug mode. • **Hands-on:** Create a virtual environment, install Flask, scaffold app.py with Copilot, run server and display “Hello World” page.	**1.5 hrs**

Building Flask Application Structure	• **Routes & Blueprints** - Create multiple routes with Copilot, introduction to GET vs POST routes. • **Templates with Jinja2** - Use Copilot to generate HTML template (base layout with Bootstrap). Create Home, About pages. • **Static files** - Serving CSS and JS via Flask's static folder, using Copilot to scaffold stylesheet and link in templates. • **Prompting strategies** - Descriptive prompts for Copilot, few-shot examples inside app.py to generate consistent routes/templates. • **Hands-on:** Scaffold Home, About, and Contact routes with Copilot; generate corresponding templates; create a basic Bootstrap navbar for navigation.	**2 hrs**
Consuming JokeAPI in Flask	• **API basics** - Introduction to JokeAPI (https://v2.jokeapi.dev), explore endpoints (random jokes, categories, single vs two-part jokes). • **Requests module** - Use requests library to fetch data, parse JSON, display in Flask template. Copilot prompts to generate boilerplate for API calls. • **Dynamic routes** - Build /joke route to fetch random joke, /joke/<category> to fetch jokes by category. • **Error handling** - Handling network errors, invalid categories, empty responses use Copilot to scaffold try/except and error messages in template. • **Hands-on:** Use Copilot to generate a get_joke() function that fetches from JokeAPI and displays in /joke page. Add dropdown filter (Programming, Misc, Dark jokes). Show jokes dynamically in Bootstrap card UI.	**2 hrs**
Enhancing Flask App with Copilot	• **Forms & User Input** - Use Flask-WTF or plain forms for user to select joke type (single/two-part), let Copilot scaffold form validation and CSRF protection. • **Refactoring** - Move API calls into a separate services/joke_service.py file; use Copilot to generate helper functions and reuse logic. • **Testing with pytest** - Intro to pytest, Copilot generates unit tests for get_joke() service and route responses, simulate API calls with mocks. • **Documentation & Logging** - Copilot adds docstrings (PEP 257 style), inline comments, and simple logging with Python logging module. • **Hands-on:** Add a "Get Custom Joke" form, refactor API logic into a service file, generate 2–3 pytest test cases with Copilot, and add docstrings for main functions.	**1.5 hrs**

Wrap-up & Best Practices	• **Production readiness** - Flask app structure (app factory pattern intro), environment variables for secrets (Flask Config class), Copilot for generating .env and config management. • **Security basics** - Sanitize user input, avoid exposing API keys, use environment configs. Copilot suggestions for input validation. • **Responsible Copilot usage** - Validating Copilot output, avoiding blind trust, copyright/licensing of generated code. • **Final showcase** - Polish templates with Bootstrap, run tests, final demo of JokeAPI Flask app. • **Hands-on:** Deploy-ready build (flask run), run tests, present the JokeApp with category-based and random jokes.	**1 hr**

Learning Pedagogy

The pedagogic model is largely focused on experiential learning in remote virtual learning mode. Some expert mentors shall work with students through the program. Learning is in an environment that combines the convenience of online and anytime access with the intensity of mentoring.

The model combines the following elements:

1. Virtual Instructor-led Live connects: These work on a fixed schedule with recorded versions available to people who miss them.
    - Sessions that provide context
    - Sessions that demonstrate the usage of tools or technologies
    - Sessions that explain best practices.
    - Sessions that explain common pitfalls/issues during code reviews
2. Reference for self-study videos and learning material:
    a. Video-based material and curated sites to refer will be provided as reading.
    b. Exercises/Assignments for participants to practice and defined User stories.
3. Focused small-group tutoring: These are organized by mentors and participants are invited to participate in them based on their current needs.
4. In addition, on-demand participant support will be provided via team collaboration tools such as Slack.
5. Very intensive hands-on component with participants expected to complete the hands-on activities such as completing all activities
6. **Course Durations are flexible with part-time learning.**