Day 46 Tasks

1. Write a short note (in your own words) on what Flask is.

Flask is a lightweight, Python-based web framework designed for building simple and flexible web applications. It's known for its minimalistic design, allowing developers to choose their tools and libraries as needed. Flask gives you more control over your application's structure compared to full-stack frameworks like Django.

---

2. List 5 differences between Flask and Django in a table format.

Feature	Flask	Django
Framework Type	Microframework	Full-stack framework
Flexibility	Highly flexible, choose components	Opinionated with built-in components
Admin Interface	Not included by default	Comes with a powerful admin interface
Learning Curve	Easier for beginners	Steeper due to its many features
Project Structure	You define structure	Predefined project structure

---

3. Research and write 3 pros and 3 cons of using Flask.

Pros:

1. **Lightweight and Flexible** – Gives more control over components and project structure.
2. **Easy to Learn** – Minimalistic approach is beginner-friendly.
3. **Great for Microservices** – Flask's modularity makes it perfect for building APIs and microservices.

Cons:

1. **Lacks Built-in Tools** – Features like authentication and admin need to be manually integrated.
2. **Scalability Challenges** – Requires careful planning for larger, complex applications.

3. **Not Opinionated** – While flexible, it may lead to inconsistent coding practices without strong guidelines.

---

4. Explain with examples which types of projects are best suited for Flask.

Flask is ideal for:

- **APIs and Microservices:** REST APIs, like a weather data API or a news feed backend.
- **Prototypes/MVPs:** Quick idea testing, like a prototype of a travel booking site.
- **Small Web Apps:** Portfolio websites, blogs, or TODO apps.

Example: A startup building a basic user feedback platform can use Flask for quick development and later scale or migrate if needed.

---

5. Compare Flask and Django based on project setup complexity.

- **Flask:** Very simple setup. You can build and run a "Hello, World" app in a few lines. You choose all additional libraries.
- **Django:** More complex setup due to built-in features like ORM, admin, templating, and authentication. Comes with a predefined project layout.

Conclusion: Flask has a much lower initial setup complexity compared to Django.

---

6. Describe the role of WSGI in Flask applications.

WSGI (Web Server Gateway Interface) is a specification that acts as a bridge between web servers and Python web frameworks. In Flask:

- Flask is a **WSGI-compliant** application.
- When a request comes in, the web server (e.g., Gunicorn, uWSGI) passes it to Flask through WSGI.
- Flask processes the request and sends back a response via WSGI to the server.

---

7. List and explain 5 companies or platforms that use Flask in production.

Company/Platform	Usage
Netflix	Uses Flask for internal APIs and automation tools

Company/Platform	Usage
Airbnb	Some backend microservices are built using Flask
Reddit	Utilizes Flask for smaller services and tooling
Uber	Flask is used in various internal services due to its simplicity
MIT	Flask is used in educational platforms and academic tools

---

8. Create a mind map comparing Flask and Django's architecture.

Flask vs Django Architecture

Flask

|

├── Routing (Flask.route)

├── Templates (Jinja2)

├── No ORM by default

├── Extensible with third-party packages

└── WSGI-compatible

Django

|

├── URL Dispatcher (urls.py)

├── Templates (Django templating engine)

├── Built-in ORM (models.py)

├── Admin Panel, Forms, Auth built-in

└── Follows MTV (Model-Template-View)

---

9. List reasons why Flask is preferred for beginners.

1. **Minimal Setup:** Easy to start with a single file.

2. **Less Abstraction:** Helps beginners learn how things work under the hood.
3. **Documentation:** Clear and beginner-friendly.
4. **Flexible Structure:** You're not forced into a specific project architecture.
5. **Great Community and Tutorials:** Lots of beginner resources available.

---

10. Choose a sample use-case (like a TODO app) and explain why Flask would be a better choice than Django.

Use-Case: Building a **TODO list application**.

Why Flask is better:

- The app is simple and doesn't need Django's full suite of tools.
- You can get started with just a few lines of code.
- No need for an admin interface or complex ORM features.
- Faster development cycle and less overhead.
- Easier for learning the basics of routing, forms, and templating.

Example: You can build and deploy a TODO app in Flask with fewer than 100 lines of code.

---

11. Create a virtual environment and activate it

python -m venv venv

venv\Scripts\activate

---

12. Install Flask using pip and verify the installation

Command:

pip install Flask

Verify Installation:

python -m flask –version

---

13. Check the installed Flask version and print it

python -m flask --version

14. Create a requirements.txt file with Flask listed

Command:

pip freeze > requirements.txt

---

15. Install Flask from a requirements.txt file in a new environment

python3 -m venv newenv

source newenv/bin/activate # or newenv\Scripts\activate (Windows)

pip install -r requirements.txt

---

16. Upgrade Flask to the latest version using pip

pip install --upgrade Flask

---

17. Uninstall Flask and reinstall it

Uninstall:

pip uninstall Flask

Reinstall:

Pip install Flask

---

18. Explore and list down any 5 Flask-related packages on PyPI

Package Name	Description
Flask-Login	User session management and authentication
Flask-WTF	Integrates WTForms with Flask for form handling and CSRF
Flask-Migrate	SQLAlchemy database migrations using Alembic
Flask-Mail	Sending emails from Flask applications
Flask-CORS	Enables Cross-Origin Resource Sharing for Flask apps

---

19. Use pip freeze to generate a complete environment dependency list

pip freeze > requirements.txt

---

20. Create a bash script to set up a virtual environment and install Flask automatically

File: setup_flask.sh

#!/bin/bash

 # Create virtual environment

python3 -m venv venv

 # Activate environment

source venv/bin/activate

 # Upgrade pip

pip install --upgrade pip

 # Install Flask

pip install Flask

 # Confirm Flask installed

python -m flask –version

---

29. Create an app.run(debug=True) statement and explain the debug mode

debug=True does 2 things:

- **Auto-reload:** Automatically restarts the server when code changes.
- **Debug Error Page:** Shows detailed error messages if something breaks.

💡 **Note:** Don’t use debug=True in production – it exposes sensitive info.

---

32. Use the --debug flag while running the server and observe the behavior

flask --app app.py --debug run

 Enables **debug mode** (auto-reload + error debugger).

 You'll see: Debugger is active!

---

36. Run the Flask app in production mode (not using debug)

In app.py

Remove debug = True

Then command:

Set FLASK_ENV=production

 # debugger is disabled

To check run: python app.py

Output: debugger: off -its showing

---

37. Explain the difference between running with python app.py and flask run

Feature	**python app.py**	**flask run**
Debug auto-reload	Manual (debug=True in code)	Built-in if --debug is used
App discovery	Needs if __name__ == '__main__'	Uses FLASK_APP variable
Simplicity	Quick and explicit	Cleaner, uses Flask CLI
Production readiness	Not recommended	Still not for production, but more flexible

---

40. Write a note on common issues faced while running the server

Issue	Description	Fix
Port Already in Use	Another app (or old Flask instance) is using port 5000	Use another port: --port=8000
Environment Variable Missing	FLASK_APP not set	Set it: export FLASK_APP=app.py
Firewall Blocking	Accessing Flask from another device	Allow port in firewall settings
Syntax Errors	Mistyped Python code	Check traceback in debug page
Debug Not Reloading	Auto-reload not working	Ensure --debug is used or debug=True in code

---

45. Explain what happens when you define two functions with the same route

This example is not allowed.

```
@app.route('/conflict')
def first():
    return "First function"


@app.route('/conflict')
def second():
    return "Second function"
```

only second function will be used.

It causes confusion.