🟢Beginner Level

1. **What is Flask and How It Works**
2. Flask is a lightweight Python web framework used to build web applications. It handles routing, templates, and HTTP requests.
3. **Installing Flask and Setting Up a Virtual Environment**
4. Use `venv` to create isolated Python environments and install Flask with `pip install flask`.
5. **Your First Flask App: Routing and Views**
6. Create a simple app with `@app.route()` to define routes and functions that return responses.
7. **Flask Debug Mode and Development Server**
8. Enable `debug=True` to auto-reload on changes and show error messages in development.
9. **HTTP Methods: GET and POST**
10. Understand how browsers use GET for retrieving data and POST for sending data (e.g., forms).
11. **Templates with Jinja2**
12. Use HTML templates and insert dynamic content with Jinja2 syntax like `{{ name }}`.
13. **Rendering Dynamic HTML with Jinja2**
14. Pass data from your Flask app to templates using `render_template()`.
15. **Working with Forms and Request Data**
16. Use `request.form` or `request.args` to handle form data sent by the user.
17. **Serving Static Files (CSS, JS, Images)**
18. Place static assets in a `static/` folder and link them in templates using `url_for('static', filename='...')`.

🟡Intermediate Level

1. **Flask Project Structure (Modular Apps)**

2. Organize your app with folders like `templates/`, `static/`, and `routes.py` for maintainability.
3. **Flask Blueprints for Scalable Apps**
4. Use blueprints to separate different parts of the app (e.g., auth, admin) into reusable components.
5. **Using WTForms and Form Validation**
6. Create forms in Python and validate input with Flask-WTF.
7. **Redirects and URL Building**
8. Use `redirect()` and `url_for()` to navigate between routes programmatically.
9. **Flask Session and Cookies**
10. Store data between requests using `session` (e.g., login info) or set cookies.
11. **Using Flask with SQLite or PostgreSQL**
12. Connect your app to a database to store persistent data.
13. **Connecting to Databases using SQLAlchemy**
14. Use SQLAlchemy ORM to interact with databases using Python classes.
15. **Creating Models and Relationships with SQLAlchemy**
16. Define tables and relationships (one-to-many, many-to-many) as Python classes.
17. **Flask CLI and Shell Context**
18. Extend the CLI for custom commands and access app context from the terminal.
19. **Flash Messages for User Feedback**
20. Use `flash()` to show success or error messages after actions (e.g., login).

🔹 Authentication & Authorization

1. **User Registration and Login System**
2. Build user sign-up and sign-in functionality with sessions.

3. **Password Hashing with Werkzeug**
4. Securely hash and verify passwords using `generate_password_hash()` and `check_password_hash()`.
5. **User Sessions and Login Management**
6. Keep users logged in using Flask sessions.
7. **Protecting Routes with Login Required**
8. Block access to certain pages unless the user is logged in.
9. **Using Flask-Login for User Management**
10. A package that simplifies login/logout and managing user sessions.
11. **Role-based Access Control (RBAC)**
12. Grant or restrict access based on user roles (admin, user, etc.).

🔸 Advanced Level

1. **RESTful API Development with Flask**
2. Build APIs that return JSON and follow REST principles (GET, POST, PUT, DELETE).
3. **Flask Marshmallow for JSON Serialization**
4. Convert models to JSON and validate input using Marshmallow.
5. **Flask-Migrate for Database Migrations**
6. Automatically generate and apply changes to your database schema.
7. **Testing Flask Applications (PyTest / unittest)**
8. Write tests to ensure your app works as expected.
9. **Error Handling and Custom Error Pages**
10. Handle errors gracefully and show user-friendly error pages (404, 500, etc.).
11. **Using Flask-RESTful or Flask-Smorest**

12. Build modular and documented REST APIs easily.
13. **File Uploads and Handling**
14. Allow users to upload files and handle them securely.
15. **Pagination and Filtering**
16. Manage large datasets with page-by-page viewing and search filters.
17. **Caching with Flask-Caching or Redis**
18. Improve performance by storing frequently used data temporarily.
19. **Rate Limiting (Flask-Limiter)**
20. Prevent abuse by limiting how often users can access endpoints.

🛑Deployment and DevOps

1. **Environment Variables and Configuration Management**
2. Store sensitive or environment-specific settings safely.
3. **Logging and Monitoring Flask Apps**
4. Track errors and logs for debugging and performance insights.
5. **Deploying Flask on Render / Heroku / Vercel**
6. Push your app live using platforms that support Python.
7. **Deploying Flask with Gunicorn and Nginx (Linux VPS)**
8. Use production-ready tools to serve your Flask app on a VPS.
9. **Using Docker with Flask**
10. Containerize your app to run it consistently across environments.
11. **Flask with GitHub Actions (CI/CD)**
12. Automate testing and deployment with GitHub Actions workflows.

🌐Optional but Useful

1. **Using Flask with Frontend Frameworks (Vue, React)**
2. Integrate a modern JavaScript frontend with Flask as the backend.
3. **JWT Authentication with Flask-JWT-Extended**
4. Use JSON Web Tokens to authenticate users in APIs.
5. **WebSockets with Flask-SocketIO**
6. Build real-time apps like chat or live notifications.
7. **Task Queue with Celery and Flask**
8. Run background tasks (like sending emails) asynchronously.
9. **Admin Interfaces using Flask-Admin**
10. Add a backend admin panel for managing data easily.