□ 1. FOUNDATIONAL KNOWLEDGE

□ Programming Basics

Before diving into Python, understand these fundamentals:

- What is a programming language?
- What are variables, data types, and operators?
- How programs execute (compilation vs interpretation)

□ Learn Python (Core)

Key Topics:

- Variables, Data Types, Type Casting
- Input/Output
- Operators
- Conditional Statements (`if`, `elif`, `else`)
- Loops (`for`, `while`)
- Functions (arguments, return values, default params)
- Lists, Tuples, Dictionaries, Sets
- String Manipulation
- Exception Handling
- File Handling (read/write)
- Modules and Packages
- Virtual Environments (`venv`, `pip`)

Recommended Practice:

- LeetCode easy Python problems (strings, arrays)
- Write small programs like:
  - Calculator
  - To-do list (CLI)
  - Number guessing game

□ 2. PYTHON ADVANCED

Key Topics:

- Object-Oriented Programming (OOP)
  - Classes, Objects
  - Inheritance, Polymorphism
  - Encapsulation, Abstraction
- Lambda, Map, Filter, Reduce
- List Comprehension
- Decorators & Generators
- Modules: `os`, `sys`, `json`, `datetime`
- Error handling best practices
- Logging (`logging` module)

Mini Projects:

- Student Management System (OOP based)
- Expense Tracker CLI app (files + classes)
- JSON-based data storage

□ 3. DATABASE & DATA HANDLING

SQL (MySQL/PostgreSQL):

- Create, Read, Update, Delete (CRUD)
- Joins, Primary/Foreign Keys
- Aggregate functions
- Indexing, normalization

Python Integration:

- Use `mysql-connector` or `psycopg2`
- Execute CRUD operations from Python

NoSQL (MongoDB):

- Documents, Collections

- CRUD using `pymongo`

Mini Projects:

- Contact Book (SQLite)
- Student Result Portal (PostgreSQL)

□ 4. VERSION CONTROL (GIT & GITHUB)

Key Topics:

- `git init`, `add`, `commit`, `status`, `log`
- Branching (`git branch`, `checkout`, `merge`)
- Remote Repos (GitHub)
- Pull Requests & Forks
- Resolving merge conflicts
- `.gitignore` usage
- GitHub Actions (intro to CI/CD)

Mini Project:

- Create a public GitHub repo
- Push your Python mini projects
- Create a README with installation steps

□ 5. FRONTEND DEVELOPMENT

Even backend devs must understand frontend basics.

HTML:

- Structure, tags, links, tables, forms

CSS:

- Layouts (Flexbox, Grid)
- Colors, Fonts, Animations

- Responsive design (media queries)

JavaScript:

- Variables, Functions, Events
- DOM Manipulation
- Fetch API (GET/POST requests)
- Async/Await, Promises

Frontend Framework (optional for Python full stack):

- React.js (if you want MERN-style)
- OR use HTML + Bootstrap/PrimeFlex for simple projects

Mini Projects:

- Portfolio website
- Responsive login/signup page

□ 6. BACKEND DEVELOPMENT WITH PYTHON

□ Frameworks:

□ *Flask (Lightweight)*

- Routing
- Templates (Jinja2)
- Request/Response
- REST API endpoints
- JSON responses
- Connecting Flask with SQL (SQLAlchemy)

OR

□ *Django (Full-Stack Framework)*

- MTV architecture
- Models, Views, Templates

- ORM
- Django Admin
- Authentication & Authorization
- REST Framework (Django REST Framework)

Concepts to Learn:

- REST API design
- CRUD APIs
- Authentication (JWT, session)
- Error handling
- Logging
- Middleware

Mini Projects:

- Blog API (Flask or Django)
- Todo App API
- Login/Register system (JWT-based)

□ 7. API DEVELOPMENT & INTEGRATION

Learn:

- REST API fundamentals (GET, POST, PUT, DELETE)
- JSON request/response
- API documentation using Swagger/Postman
- Testing APIs (Postman, pytest)
- Authentication:
  - JWT tokens
  - OAuth (Google login)
- Rate Limiting & Pagination

Mini Projects:

- Weather API (fetch data from OpenWeather API)
- Currency Converter API
- Email OTP Verification API

□ 8. FRONTEND + BACKEND INTEGRATION (Full Stack)

Goal: Connect your Python backend (Flask/Django) with frontend (HTML/JS/React).

Flow:

1. Create REST API in Flask/Django.
2. Use Fetch or Axios in frontend to call backend APIs.
3. Display data dynamically.

Project Ideas:

- Expense Tracker (Frontend + Flask backend + SQLite)
- Task Manager (React + Django REST)
- Notes App (CRUD + Auth + API Integration)

□ 9. DEPLOYMENT & CLOUD BASICS

Learn:

- Hosting Flask/Django apps
  - Render / Railway / Vercel / AWS EC2
- Database hosting (AWS RDS, MongoDB Atlas)
- Environment Variables
- Docker (optional)
- Nginx / Gunicorn basics

Mini Projects:

- Deploy your full stack app on Render
- Connect to hosted database

□ 10. TOOLS EVERY DEVELOPER MUST KNOW

Tool	Purpose	Learn
VS Code	IDE	Extensions, shortcuts
Postman	API testing	Collections, environments
Git & GitHub	Version Control	Branching, pull requests
Docker (Optional)	Containerization	Dockerfile, compose
Linux Commands	CLI proficiency	`cd`, `ls`, `grep`, `vim`
CI/CD (Basic)	Automation	GitHub Actions
Swagger / Redoc	API Docs	Integrate in Flask/Django

□ 11. FINAL END-TO-END PROJECT IDEAS

□ Beginner

- Personal Portfolio (HTML, CSS, Flask)
- To-do App (CRUD + SQLite)
- Weather App (API integration)

□ Intermediate

- Expense Tracker (React + Flask/Django + SQL)
- Blog System with Auth (JWT, CRUD)
- Notes Sharing App with file upload

□ Advanced

- Learning Management System (Django REST + React)
- E-commerce (Django backend + React frontend + Stripe payments)
- Chat App (WebSocket + Flask-SocketIO)
- AI-powered Resume Analyzer (Flask + OpenAI API)