📘 AI/ML Software Foundations Curriculum (Oct–Dec)

Notes

- You can use **any IDEs** you like (VS Code, PyCharm, Jupyter, etc.).
- You are free to use **AI assistants / chatbots** (e.g., Gemini, ChatGPT, Copilot) during learning.
- ⚠️ For **projects**, you are strongly encouraged to **implement solutions yourself** → this is the only way to build real engineering confidence.
- Document your code well and follow Git best practices for every project.

Oct Week 1–2: Python Foundations

Why this matters

Python is the backbone of AI/ML development. While you may already use it for model building, **production AI microservices** require:

- Clean, efficient, maintainable code.
- OOP for structuring applications.
- Error handling for robust APIs.
- File I/O for loading/saving models, configs, and logs.

Without solid Python skills, scaling from "Jupyter Notebook ML" → "deployed AI microservice" becomes extremely difficult.

Topics

- Python execution model (how code runs in CPython).
- Syntax: data types, control flow, loops, comprehensions.
- Functions, scope, *args, **kwargs.
- OOP: classes, inheritance, polymorphism.
- Exception handling & logging.
- File handling & context managers.
- Python modules, imports, and packaging.

References

- Python Basics: YouTube
- OOP In-Depth: YouTube

Project 🛗 Elevator Management System

Simulate multiple elevators in a building:

- Handle floor requests.
- Assign optimal elevator.
- Track states (idle, moving, doors open).

Oct Week 3–4: Concurrency + Gemini SDK

Why this matters

AI/ML microservices often need to **handle multiple users** simultaneously:

- Sync code → simpler, but blocks during IO (e.g., API call, file save).
- Async code → better for high-throughput systems like chatbots, streaming APIs.

Also, **LLMs (Gemini, GPT, etc.)** are accessed via APIs. You must learn:

- Async for efficient parallel requests.
- Gemini SDK to serve AI models via production APIs.

Topics

- Sync vs Async programming.
- async/await, event loop, asyncio.gather.
- Use cases: CPU-bound vs IO-bound tasks.
- Gemini SDK setup via Google AI Studio.
- Using Gemini for chat, Q&A, embeddings.

References

- AsyncIO: YouTube
- Gemini SDK Docs: Google

Projects

1. **Sync vs Async API Caller**
   - Fetch data from a public API (GitHub, Weather API).
   - Compare performance of sync (requests) vs async (httpx/aiohttp).
2. **Gemini SDK Playground**
   - Use Gemini chat model for Q&A.
   - Use embeddings for text similarity.

Nov Week 1: Git & Collaborative Development

Why this matters

In real AI/ML teams, **collaboration is essential**. You need Git to:

- Share and version-control your code.
- Work in teams without breaking each other's work.
- Enable CI/CD pipelines for deploying ML models.

Topics

- Git basics: init, clone, commit, push, pull.
- Branching, merging, PR workflows.
- GitHub repositories, issues, and collaboration.

References

- Git/GitHub: YouTube

Project

- Push Elevator + Async API Caller + Gemini demos to GitHub.
- Practice branching & PR workflow with teammates.

Nov Week 2: Web & FastAPI Foundations

Why this matters

AI/ML models are **served via web APIs**. To expose ML models to users, you need to:

- Understand HTTP & microservices.
- Use a modern API framework → FastAPI.
- Validate requests using Pydantic.

Topics

- How the Web works (HTTP, request/response cycle, REST).
- Microservices concepts.
- FastAPI routing, path/query parameters.
- Pydantic schemas for request/response validation.
- Dependency injection & middlewares.

References

- How Web Works: YouTube
- FastAPI + Pydantic: YouTube

Project 🌐 - To-Do API → CRUD tasks in memory, with request validation.

Nov Week 3–4: SQLAlchemy & Databases

Why this matters

AI services need **databases** for:

- Storing user data, logs, feedback.
- Storing embeddings & document indexes.
- Managing ML experiment metadata.

SQLAlchemy provides ORM (object-relational mapping) to integrate databases cleanly with FastAPI.

Topics

- SQL basics: queries, joins, transactions.
- SQLAlchemy ORM → models, sessions, queries.
- CRUD APIs with DB integration.
- Alembic migrations.
- Best practices for production apps.

References

- SQLAlchemy Basics: YouTube
- SQLAlchemy + PostgreSQL with FastAPI: YouTube

Project 🗄️

CRUD Application → domain of choice (Library, Inventory, Blog, etc.)

Dec Week 1: Docker & Deployment

Why this matters

Production AI apps are deployed using **containers** for:

- Reproducibility.
- Easy scaling.
- Multi-service apps (API + DB + vector DB).

Topics

- Docker fundamentals: images, containers, volumes, networks.
- Writing Dockerfile.
- Docker Compose for multi-service orchestration.

References

- Docker Basics: YouTube
- Docker Compose: YouTube

Project 🐳

- Dockerize CRUD App (FastAPI + DB).

Dec Week 2–3 (~20 Days): Final Capstone – RAG Chatbot

Why this matters

This project combines **all skills** into a real AI microservice:

- FastAPI backend.
- Database for document storage.
- Embeddings for RAG pipeline.
- Gemini SDK for LLM inference.
- Docker for deployment.

Requirements

- **Backend (FastAPI)**
  - Upload docs.
  - Generate embeddings & store (local or vector DB).
  - Chat API → RAG pipeline with Gemini + embeddings.
- **Frontend**
  - Auto-generate via **lovable** or **bolt.new** and integrate it with the FastAPI backend (no manual UI coding).
- **Deployment**
  - Docker Compose (backend + DB).

Deliverables

- Working RAG chatbot.
- GitHub repo with README + Docker setup.
- Demo of chatbot with user’s uploaded docs.