# 1. The RAG Component Map (The "Why")

Before the code, you must understand these 5 "pillars":

1. **Document Loaders:** The bridge between files (PDF, SQL, TXT) and Python.
2. **Text Splitters (Chunking):** Breaking a 100MB PDF into 500-word "mini-chapters" so the AI doesn't get overwhelmed.
3. **Embedding Model:** The "Translator" that turns text into math (Vectors).
4. **Vector Database:** The "Search Engine" that stores those math vectors (ChromaDB, FAISS).
5. **Chain/Retriever:** The "Logic" that grabs the right data and sends it to the LLM.

# 2. Phase A: Ingestion (Setting up the Brain)

This part happens **once**. You are building the knowledge base.

Python

```
from langchain_community.document_loaders import PyPDFLoader
from langchain.text_splitter import
RecursiveCharacterTextSplitter
from langchain_community.vectorstores import Chroma
from langchain_ollama import OllamaEmbeddings

# 1. LOAD: Read the PDF
loader = PyPDFLoader("classroom_notes.pdf")
raw_docs = loader.load()

# 2. CHUNK: Break it down
# chunk_overlap keeps the "bridge" between two chunks so
context isn't lost.
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=150,
    separators=["\n\n", "\n", " ", ""]
)
docs = text_splitter.split_documents(raw_docs)

# 3. EMBED & STORE: Move into ChromaDB
# nomic-embed-text is high-performance for local Mac (4GB
RAM)
embedding_model = OllamaEmbeddings(model="nomic-embed-text")
```

```
vector_db = Chroma.from_documents(
    documents=docs,
    embedding=embedding_model,
    persist_directory="./chroma_db"
)
print("✅ Ingestion Complete: Knowledge Base Created.")
```

## 3. Phase B: Retrieval & Generation (The Chat)

This part happens **every time** you ask a question.

Python

```
from langchain_ollama import OllamaLLM
from langchain.chains import RetrievalQA
from langchain.prompts import PromptTemplate

# 1. Setup the Retriever
# search_kwargs={"k": 3} means grab the 3 most similar chunks
from the PDF.
retriever = vector_db.as_retriever(search_kwargs={"k": 3})

# 2. Setup the LLM (Your fine-tuned Prometheus)
llm = OllamaLLM(model="prometheus-ai")

# 3. Define the Prompt (The "Instructions")
template = """
You are Prometheus AI, an expert assistant developed by
Likith Naidu.
Use the following context to answer the question.
If you don't know the answer, say you don't know based on the
documents.

Context: {context}
Question: {question}
Answer:"""

QA_PROMPT = PromptTemplate(template=template,
input_variables=["context", "question"])

# 4. The Chain (Connects everything)
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff", # "stuff" means just shove all chunks
into one prompt
```

```
    retriever=retriever,
    return_source_documents=True,
    chain_type_kwargs={"prompt": QA_PROMPT}
)

# 5. EXECUTE
question = "What is the focus of the current semester?"
result = qa_chain.invoke({"query": question})

print(f"\nAI Response: {result['result']}")
print(f"\nSource: {result['source_documents'][0].metadata}")
```

# 4. Professional "A-Z" Concepts (Mandatory Learning)

- **Semantic Overlap:** Why use `chunk_overlap`? If a sentence starts at the end of Chunk A and finishes at the start of Chunk B, the overlap ensures the meaning isn't "cut in half."
    - *Telugu:* ఒక వాక్యం సగం ఒక ముక్కలో, సగం ఇంకో ముక్కలో ఉంటే అర్థం మారిపోతుంది. అందుకే మనం overlap వాడతాము.
- **Vector Search vs Keywords:** Keyword search looks for "exact words." Vector search looks for "meaning."
    - *Telugu:* కీవర్డ్ సెర్చ్ అంటే సేమ్ పదం కోసం వెతకడం. వెక్టర్ సెర్చ్ అంటే ఆ పదం యొక్క **అర్థం (Meaning)** కోసం వెతకడం.
- **Chain Type "Stuff":** This is the simplest RAG method. It "stuffs" all retrieved documents into the prompt at once.

### Comparison: RAG 1.0 vs. RAG 2.0 (The Future)

| Feature | RAG 1.0 (What we built) | RAG 2.0 (Agentic) |
|---|---|---|
| **Search** | Basic Vector Search | Re-ranking (better sorting) |
| **Logic** | Single-step retrieval | Multi-step reasoning (Agents) |
| **Data** | PDF only | Database + Live API + Web |

Export to Sheets

Since you've already got your website up, the next logical step for a student is **Agentic RAG**, where the AI decides *if* it needs to look at the PDF or *if* it should search the web.