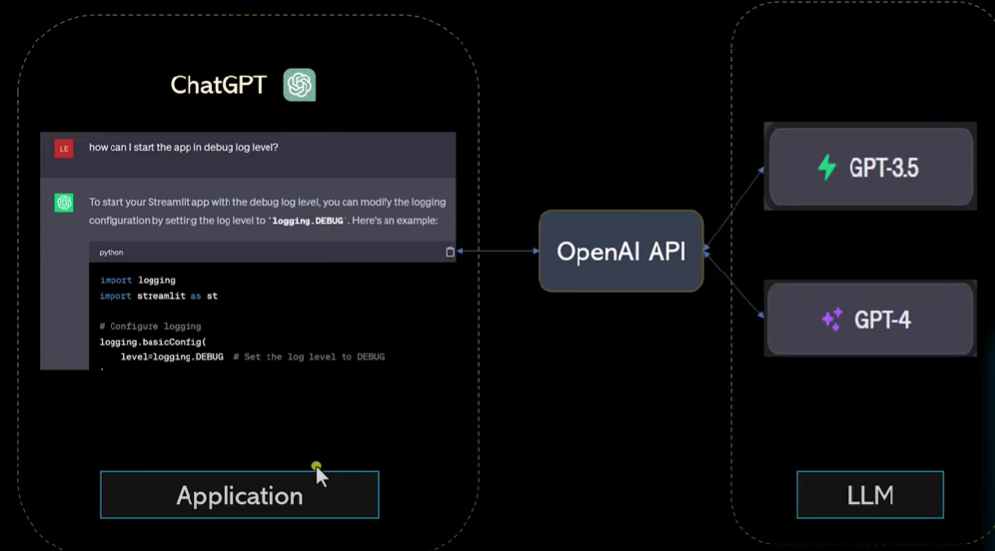
**Langchain**

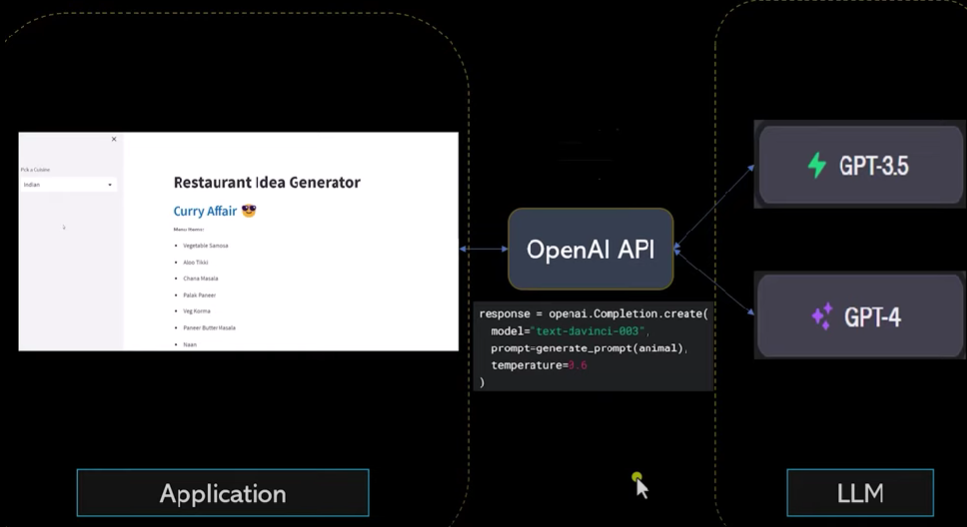
Langchain is a framework which allows you to build applications on the top of LLMs.

When we are making call to ChatGPT, it is internally making call to OpenAI api which internally using LLM such as GPT-4, GPT-3.5

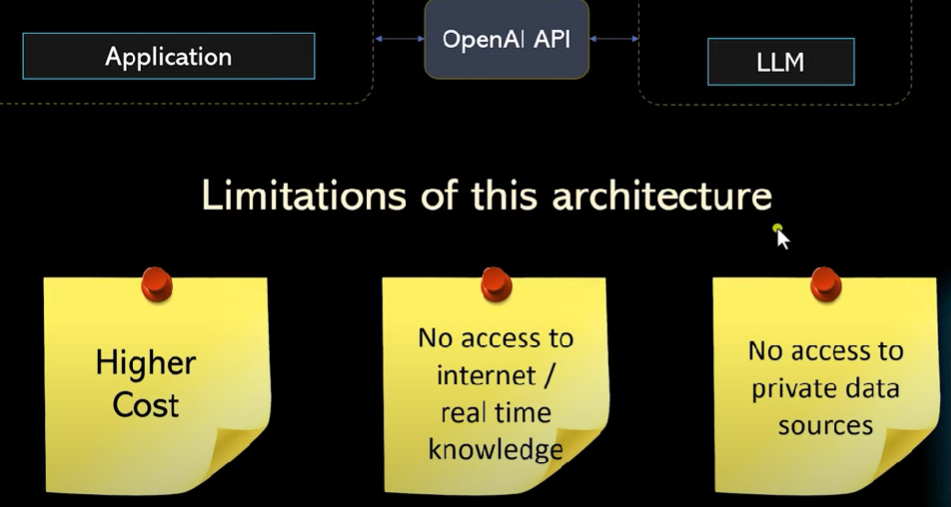
In this case ChatGPT is not an LLM, it is an application, where as GPT-3.5, GPT-4 are LLMs.



We can build an application similar kind of ChatGPT, where we can give the cuisine name and it will provide restaurant name and menu for the same.

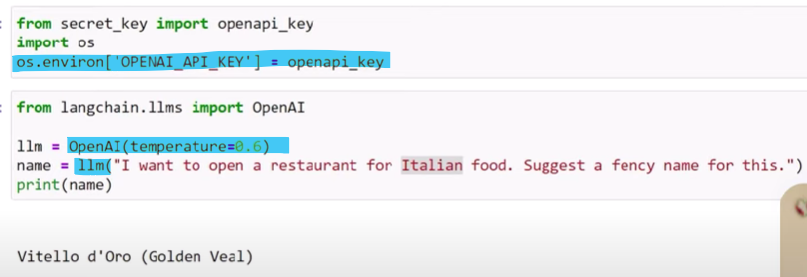


Now every business wants to build their own LLMs. Such as chatgpt because chatgpt don’t have access to their internal organization data so people want to build an application based on their llms.

Also, business not use above architecture, because calling OpenAI API has cost associated with it. Also ChatGPT doesn’t answer latest questions. As its knowledge is limited to Sept 2021. Also chatgpt doest have access to your personal organization data. 

So we can build a framework where we can use any LLM API such as OpenAI, Huggingface, Google Palm and the framework should also provide integration with google search, Wikipedia, or integration with your own organization data so that the application can pull info from the various sources. So the framework is langchain.

**Create a basic llm object**



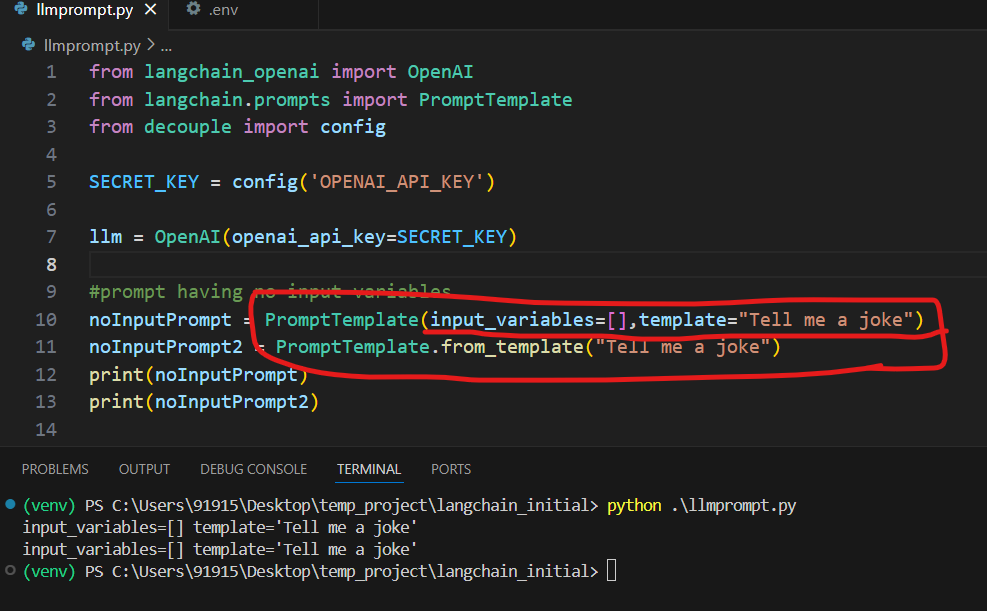
We can import any llm model from langchain.llms library such as huggingface, anthropic etc.

Temperature = 0 it means that it wont take any risk while providing output and give safe output. Where as 1 means it will try to be more creative and take risk and give the output.

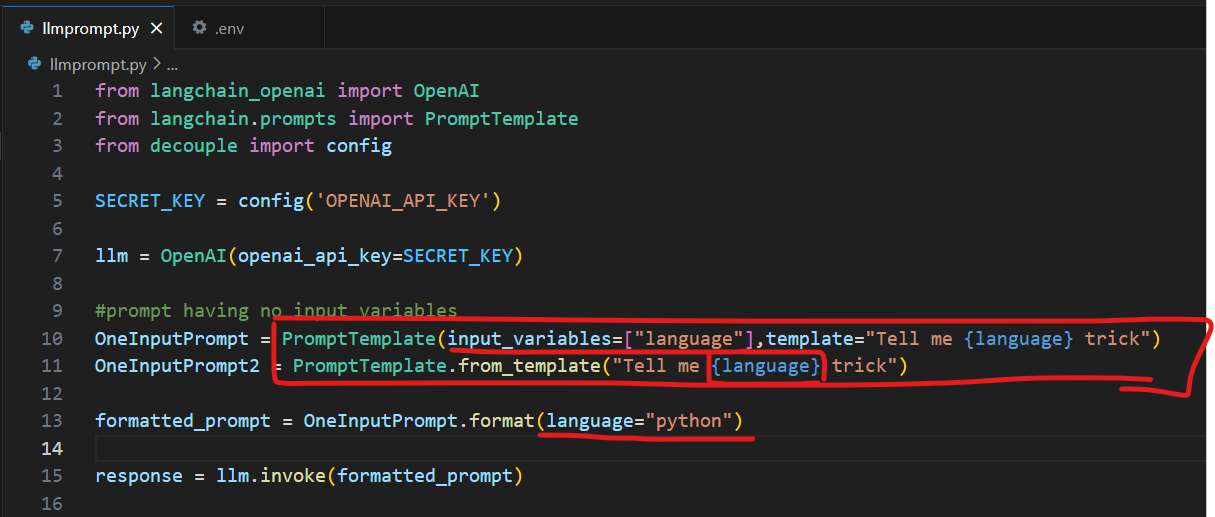
**Prompt Templates in Langchain :**

Prompt templates help to translate user input and parameters into instructions for a language model.

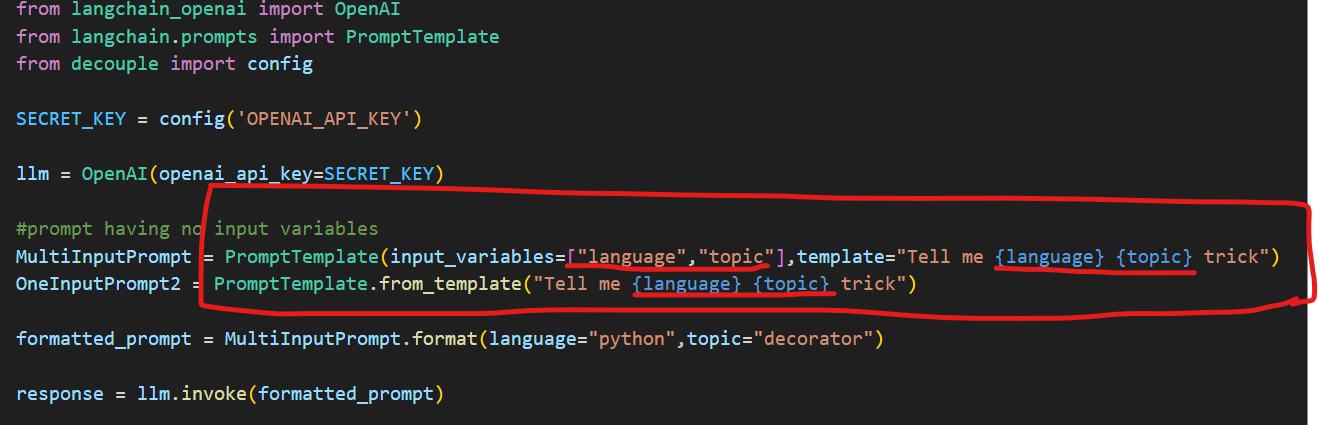
* With no input variables

****

* With one input variable.

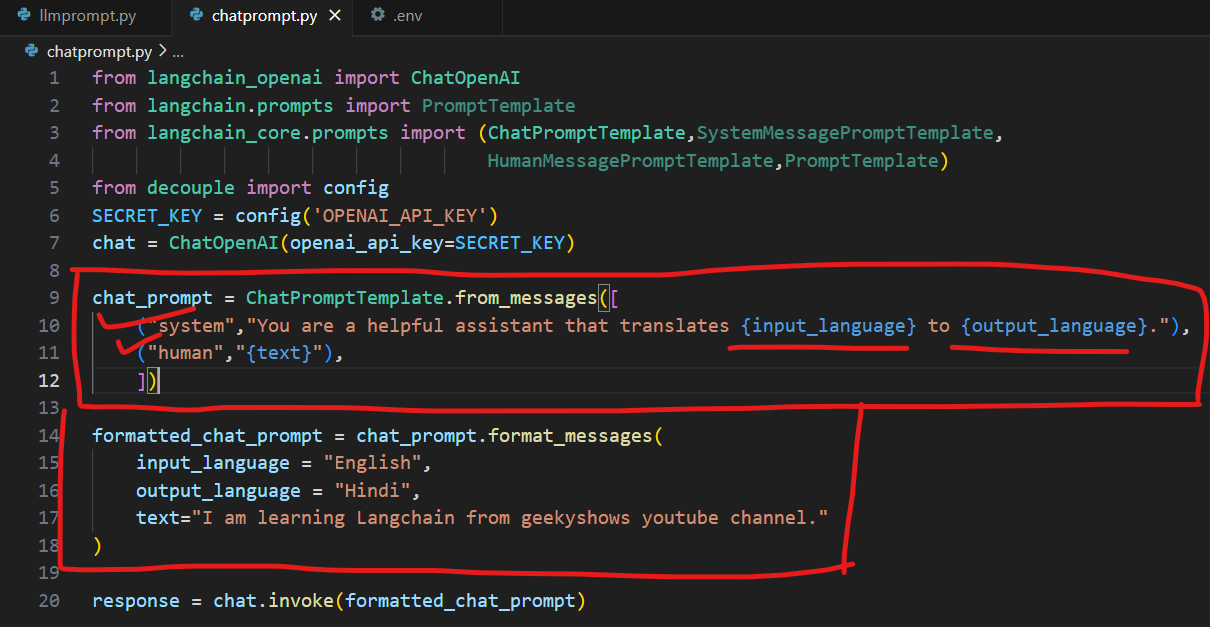


* With multiple input variables

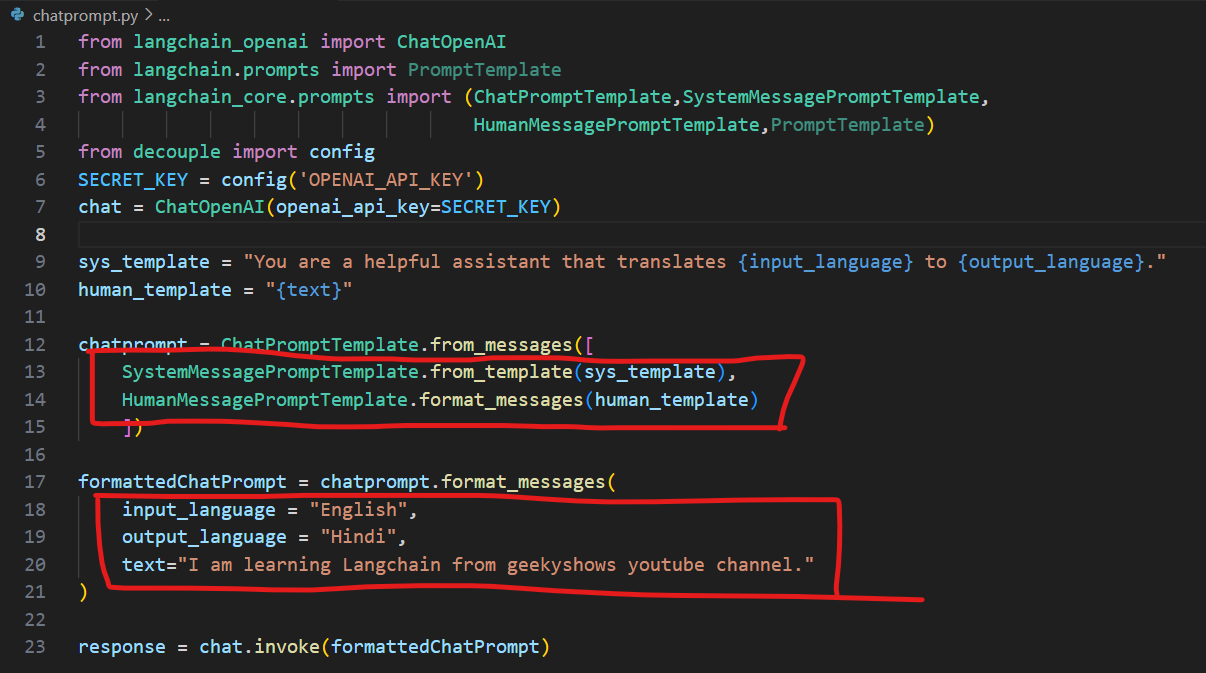


**ChatModelPrompt template**

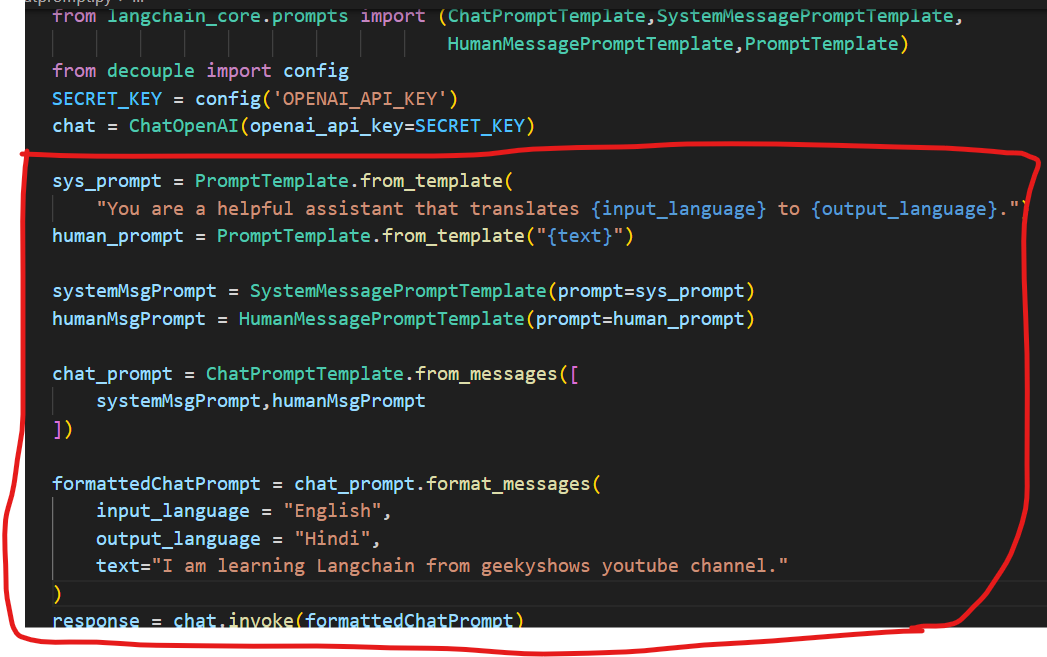
* Message prompt template as tuple



* Using message class

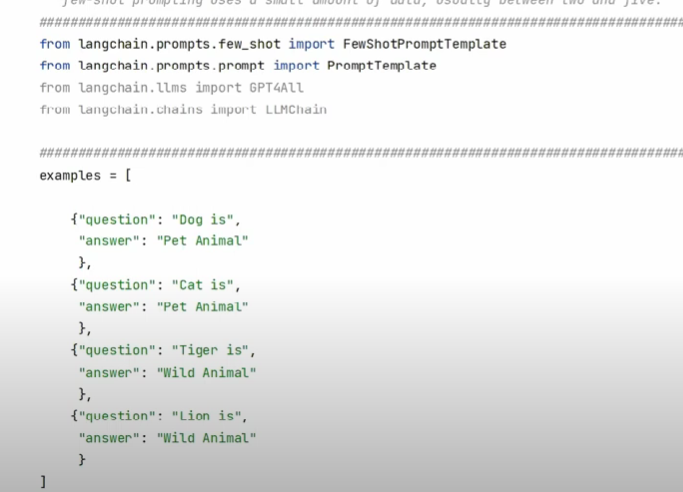


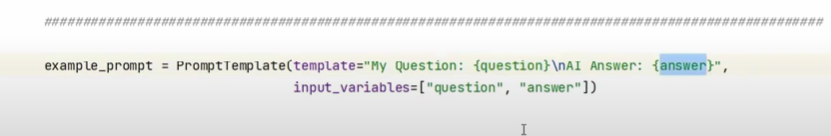
* Using PromptTemplate

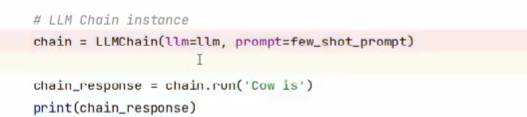
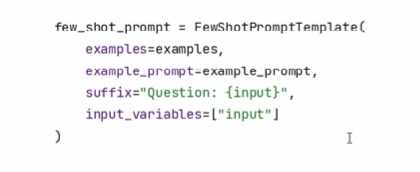


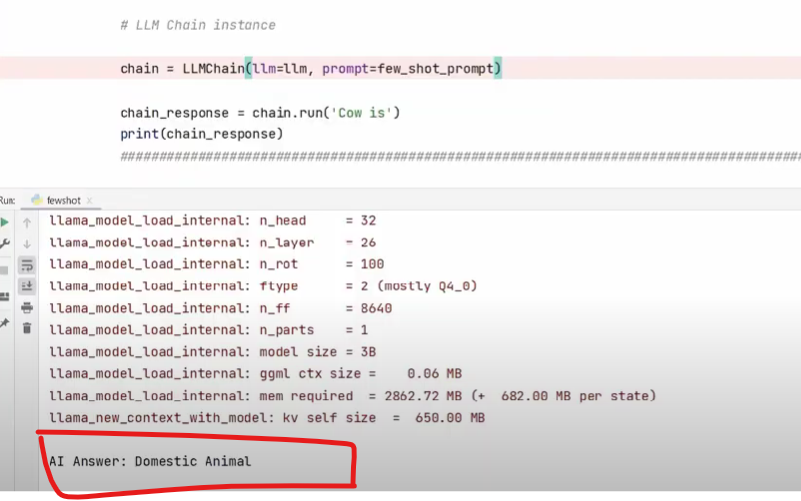
Few shot prompt template

Few shot prompting is a prompting technique which provides the Large Language Model (LLM) with a list of examples, and then asks the LLM to generate some text following the lead of the examples provided.

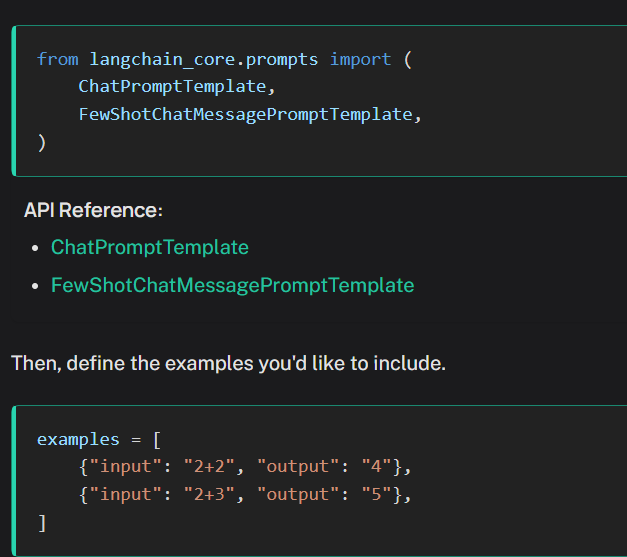




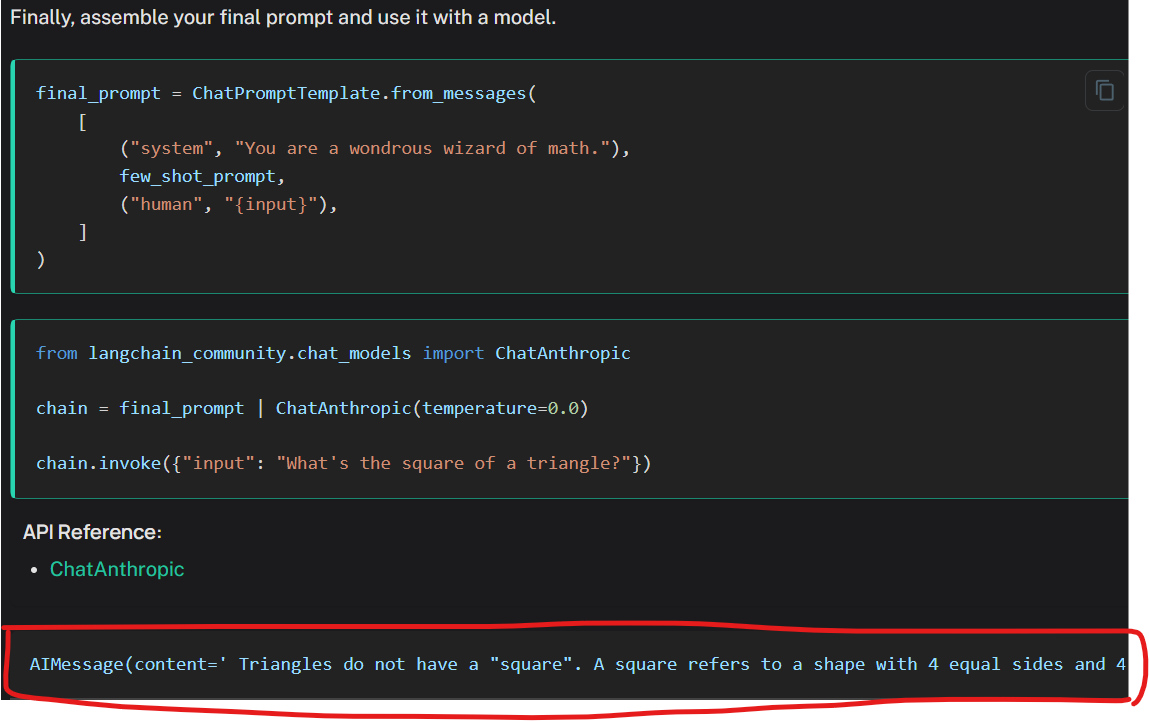




**Few-shot examples for chat models**



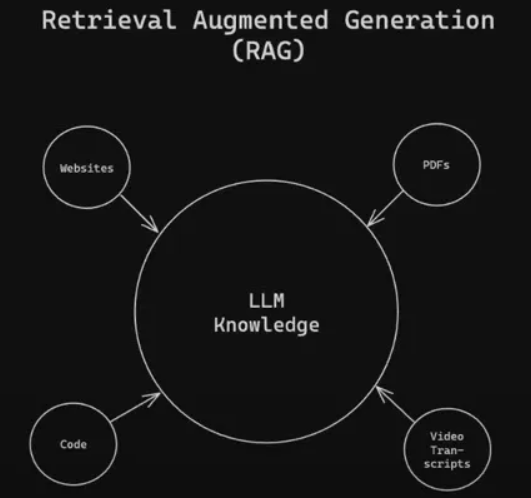




Chains: Chains refer to sequences of calls - whether to an LLM, a tool, or a data preprocessing step.

**Retrieval Augmented Generation (RAG)**

RAG is a framework that connects external sources of information with the LLM so that its responses are context-aware and accurate.



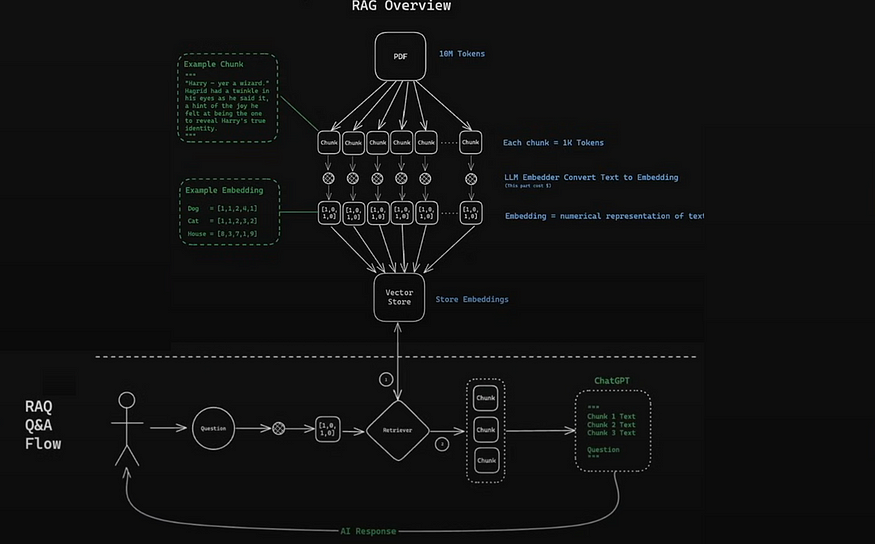
**🧠 RAG = Retrieval + Generation**

**1. Retrieval**

* Retrieves top-k most relevant documents/passages from an external knowledge base (e.g., PDF files, websites, databases, etc.).
* Uses **vector similarity** to find related chunks using embedding-based search.

**2. Generation**

* Feeds both the query and retrieved context into a **language model**.
* The model generates a response grounded in the retrieved documents.



**Input Document (PDF with 10M Tokens)**:

* The document, which contains a large number of tokens (e.g., words or characters), is broken down into smaller, manageable chunks.  Each chunk typically contains around 1,000 tokens. This makes it easier to process and analyze large texts.

**Chunking Process**:

* The large text document is divided into smaller “chunks” of text. For instance, if you have a PDF document, it could be split into several sections or paragraphs, each considered a chunk.

**Embedding Generation**:

* Each chunk is then processed by a Language Model (LLM) embedder, which converts the text into an embedding — a numerical representation of the text. This embedding captures the semantic meaning of the text and is represented as a vector (a list of numbers). For example, different words like “Dog,” “Cat,” and “House” might be converted into vectors like [1,2,4,1], [1,2,3,2], and [0,3,7,9], respectively.

**Vector Store**:

* The embeddings (numerical representations) of all the chunks are stored in a vector store. This is a database optimized for storing and retrieving high-dimensional vectors. The vector store is used to quickly find relevant chunks based on a query.

**RAQ Q&A Flow**

* **Question Input:** The process begins with a user posing a question (RAQ — Retrieval-Augmented Question Answering).
* **Retrieval Step:** The question is processed by a retriever model, which checks the stored chunks of text (from the vector store) to find the most relevant information. This step is based on similarity scoring (e.g., scores between 0.0 to 1.0).
* **Chunk Retrieval:** The retriever pulls out the most relevant chunks of text that are likely to contain the answer to the user’s question.
* **AI Processing:** These retrieved chunks, along with the original question, are passed to an AI model like ChatGPT. The AI model processes the input and generates an appropriate response.
* **Response to User:** Finally, the AI’s response is sent back to the user, completing the Q&A flow.

**Agent and Tools :**

**🧠 What Are LangChain Agents and Tools?**

**🔹 Agent:**

An Agent is an LLM-powered reasoning engine. It doesn’t just follow predefined steps like a chain — it decides what to do next based on the input.

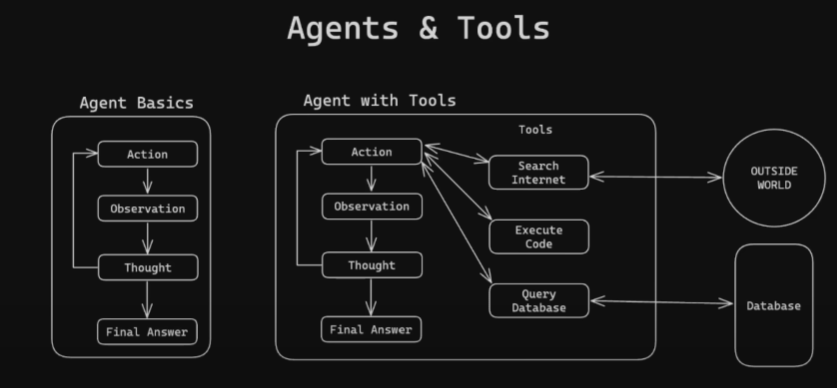
**🔹 Tools:**

Tools are custom functions you give to the agent — e.g., search engine, calculator, retriever, file reader, etc.

The Agent uses its LLM to interpret the query and decides which tools to invoke, in what order, and how to combine their results.

**🔁 Step-by-Step Execution Flow**

1. **User Input**
   * Agent receives the user query as a string.
2. **LLM Planning**
   * The agent uses its LLM (e.g., GPT) to "think" about the task.
   * It breaks down the query:
     + Get GDP of India → (maybe from search tool)
     + Compute square root → (use calculator tool)
     + Fetch summary → (use retriever tool)
3. **Tool Selection and Execution**
   * Agent chooses first tool based on the need:
     + → Uses WebSearchTool to get GDP value
   * Agent extracts GDP from the tool result
   * → Uses CalculatorTool to compute √GDP
   * Then it uses RetrieverTool to search from vector DB
   * → Summarizes the info with LLM
4. **Intermediate Reasoning**
   * After each step, agent updates its internal scratchpad (intermediate reasoning log).
   * If more steps are needed, it continues.
5. **Final Answer Construction**
   * The agent gathers all outputs, constructs the final answer, and returns it to the user.

****

We use agent and tools when we have different use case than Chains.

✅ Chains are **great when flow is known**.

✅ Agents are **better when the user might ask anything** and dynamic reasoning is needed.

✅ Tools are functions that the agent can use

**🏁 When to Use What?**

| **Scenario** | **Use Chain?** | **Use Agent?** |
| --- | --- | --- |
| Query over documents only | ✅ Yes | ❌ No |
| Chain LLM with other APIs (fixed) | ✅ Yes | ❌ No |
| User may ask math + search + docs | ❌ No | ✅ Yes |
| Need full control over steps | ✅ Yes | ❌ No |
| Need dynamic reasoning across tools | ❌ No | ✅ Yes |

**🧠 Conceptual Understanding**

| **Feature** | **Chains** | **Agents with Tools** |
| --- | --- | --- |
| **What it is** | A **fixed pipeline** of steps | A **dynamic decision-making system** |
| **Flexibility** | Static (steps are pre-defined) | Dynamic (decides next step based on user input) |
| **Control** | You control the flow | Agent controls what tool to use and when |
| **Example use** | Question → Retriever → LLM → Output | Ask anything → Agent chooses tools & answers |
| **Best for** | Simple, known tasks | Complex, open-ended queries |

**💡 Real-World Analogy**

* **Chain**: Like a **factory assembly line**. Step 1 ➜ Step 2 ➜ Step 3. Everything is predefined.
* **Agent + Tools**: Like an **AI personal assistant**. You ask it something, and it decides which "tool" to use — calculator, search, database, etc.

