Final Project Exercises

Note: These exercises are designed to give you more hands-on about langchain. In these exercises you will be exploring langchain agents, tools and how we can implement conversational memory for LLMs so that it is able to respond on our previous queries. Make changes to the code that has been provided to you by following this exercise book.

Table of Contents

Exercise # 01:

Exploring different langchain agent tools

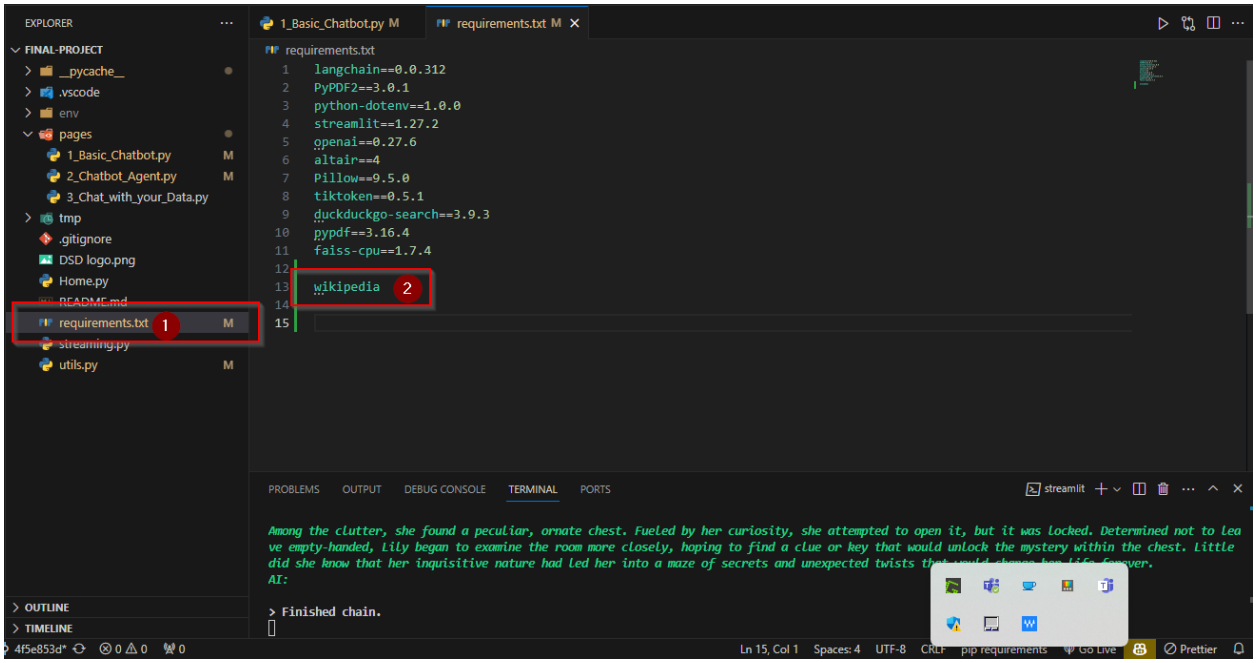
Agents are one of the most powerful and fascinating approaches to using Large Language Models (LLMs). The explosion of interest in LLMs has made agents incredibly prevalent in AI-powered use cases. Using agents allows us to give LLMs access to tools. These tools present an infinite number of possibilities. With tools, LLMs can search the web, do math, run code, and more.

In the code that has been provided to you we have used the **duckduckgosearch** tool through which you can query about recent events. In this exercise, you will be exploring the following tools:

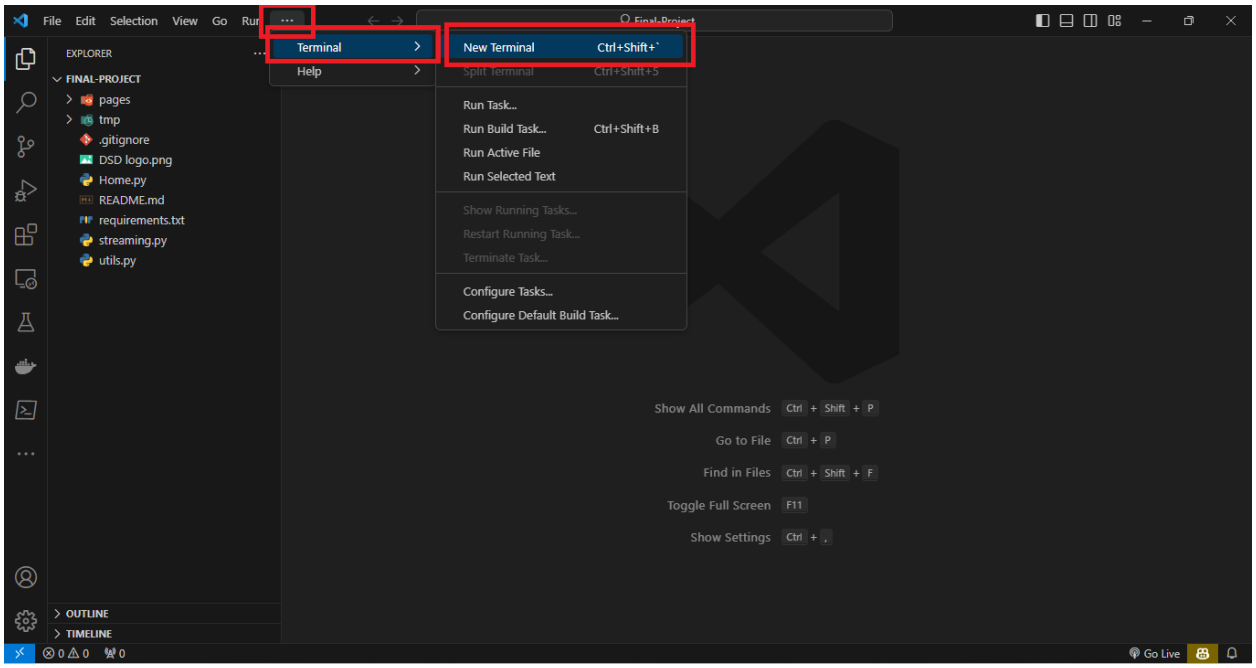
- **Wikipedia:** Ask about a specific topic, person, or event.
- **Python Repl:** Ask python codes, functions etc.

You can explore more tools from:
https://python.langchain.com/docs/integrations/tools
Follow these steps to add tools to the **Chatbot Agent**:

1. Go to **requirements.txt** in the **Final-Project** folder and add **wikipedia** to it.

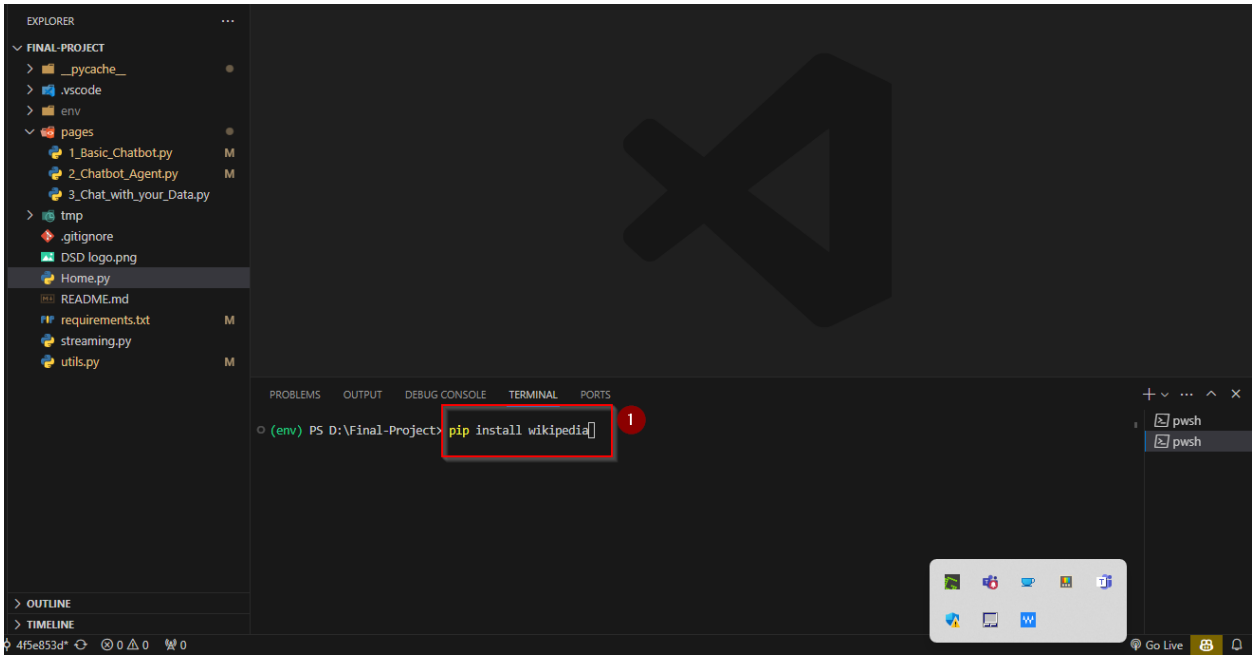


2. Open terminal in **Visual Studio Code**



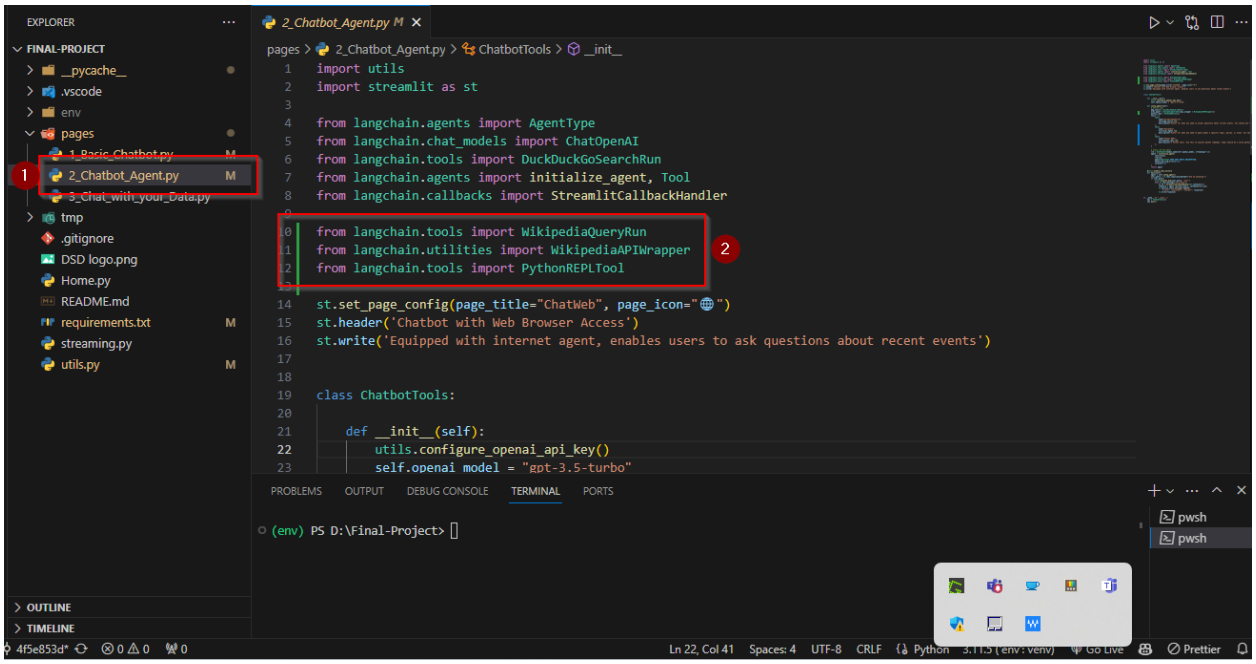
3. Run the following command in the terminal. You can **copy** the **command** from below.

   pip install wikipedia



4. Now, goto **pages** folder, open **2_Chatbot_Agent.py** and add the following imports in it. You can **copy** the **code** from below.

```
from langchain.tools import WikipediaQueryRun
from langchain.utilities import WikipediaAPIWrapper
from langchain.tools import PythonREPLTool
```

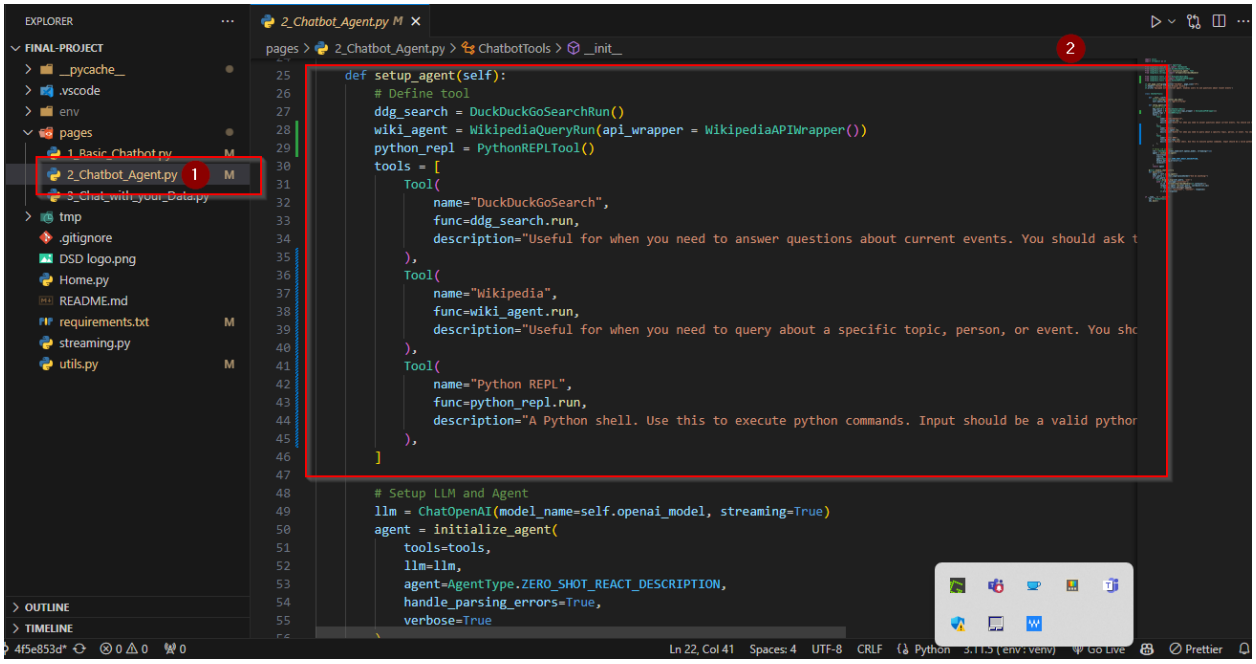


5. Now, change the code of **setup_agent** function in the same file with the following code. You can **copy** the **code** from below.

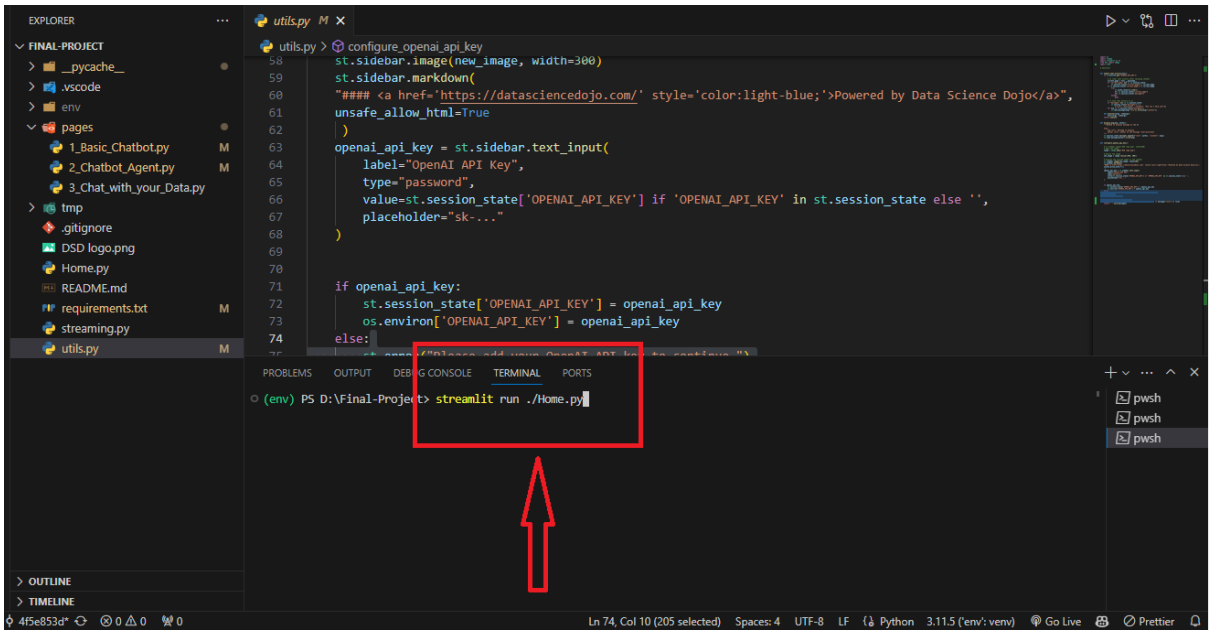
```
    def setup_agent(self):
        # Define tool
        ddg_search = DuckDuckGoSearchRun()
        wiki_agent = WikipediaQueryRun(api_wrapper =
WikipediaAPIWrapper())
        python_repl = PythonREPLTool()
        tools = [
            Tool(
                name="DuckDuckGoSearch",
                func=ddg_search.run,
                description="Useful for when you need to answer
questions about current events. You should ask targeted questions",
            ),
            Tool(
                name="Wikipedia",
                func=wiki_agent.run,
                description="Useful for when you need to query about
a specific topic, person, or event. You should ask targeted
questions",
            ),
            Tool(
                name="Python REPL",
                func=python_repl.run,
                description="A Python shell. Use this to execute
python commands. Input should be a valid python command. If you
expect output it should be printed out.",
            ),
        ]

        # Setup LLM and Agent
        llm = ChatOpenAI(model_name=self.openai_model,
streaming=True)
        agent = initialize_agent(
            tools=tools,
            llm=llm,
            agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
```

```
            handle_parsing_errors=True,
            verbose=True
        )
        return agent
```



6. Now, run the code. Open **Terminal** and write **streamlit run ./Home.py**



Exercise # 02:

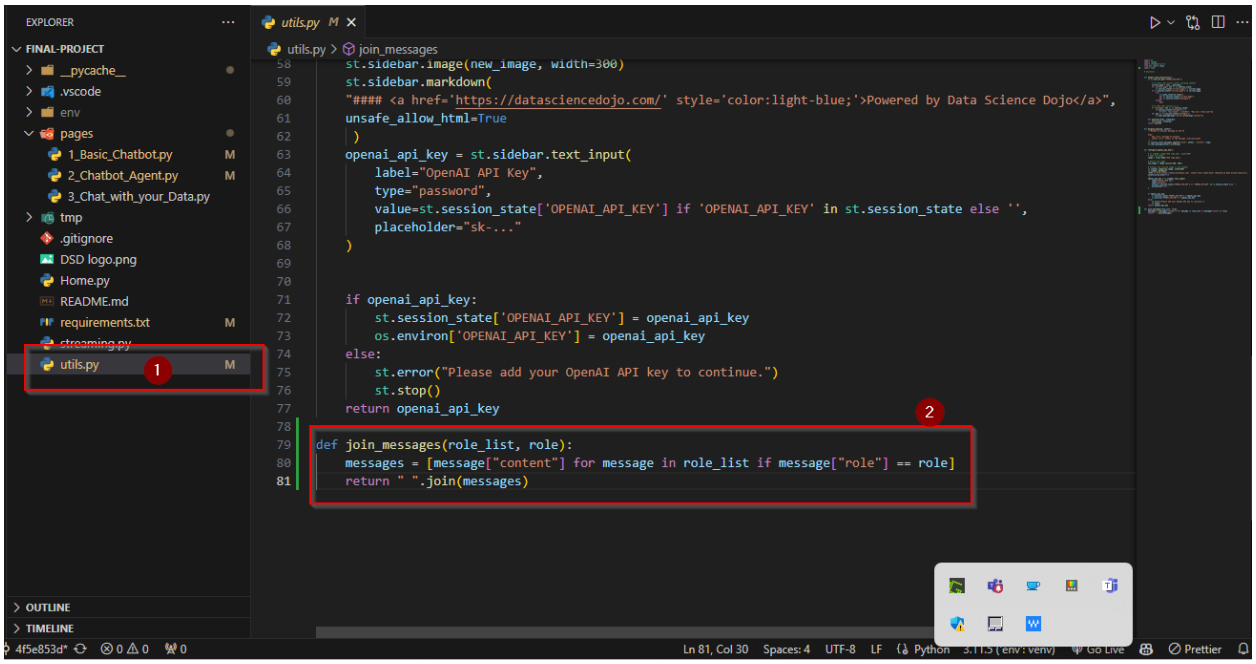
Adding Conversation Memory in Basic Chatbot

Conversation memory in the context of chatbots refers to the ability of the system to retain and utilize information from past interactions. It enables the chatbot to have a more context-aware and coherent conversation with the user. After completing this exercise you would witness that the **Basic Chatbot** would be able to respond on your previous responses

Follow these steps to add the memory in the **Basic Chatbot**:

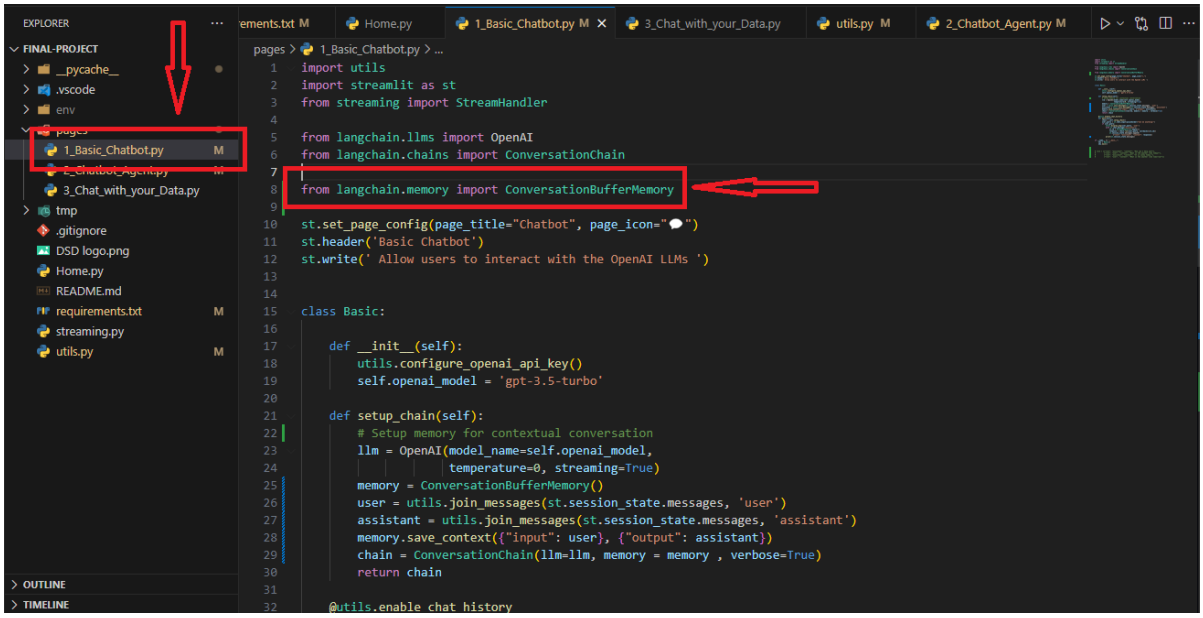
7. Open the **Final-Project** folder from the **desktop** with **Visual Studio Code** and go to utils.py and add the following code to the end of it. You can **copy** the **code** from below.

```
def join_messages(role_list, role):
    messages = [message["content"] for message in role_list if
message["role"] == role]
    return " ".join(messages)
```



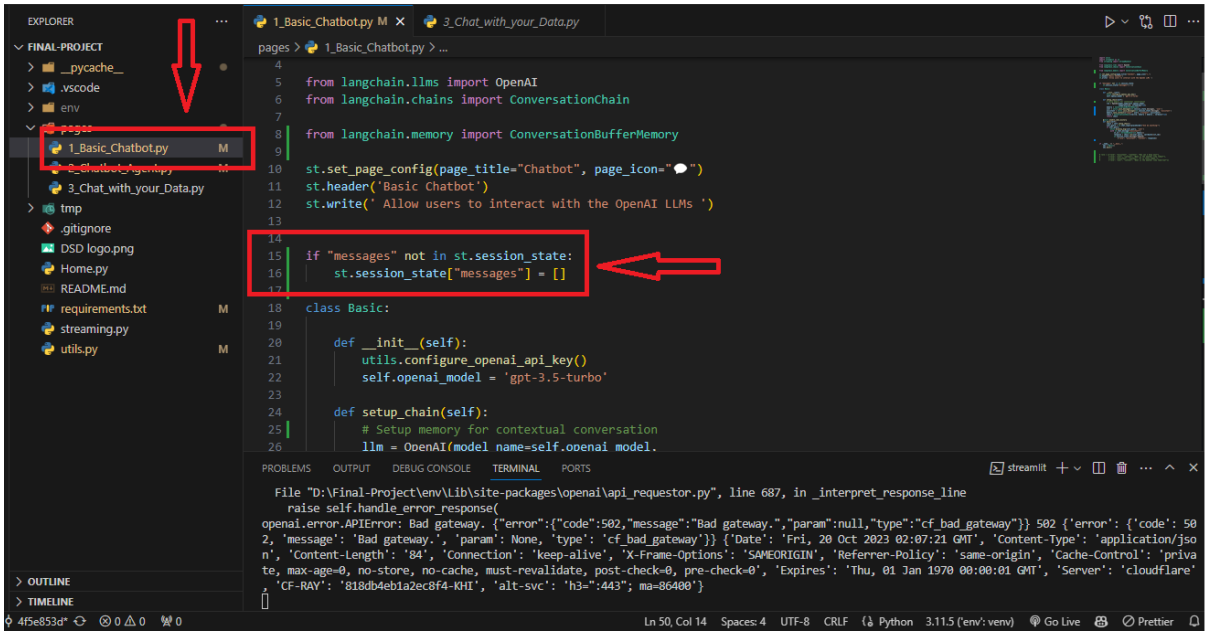
8. Now, goto **pages** folder and open **1_Basic_Chatbot.py** and add the following imports in it. You can **copy** the **code** from below.

```
from langchain.memory import ConversationBufferMemory
```



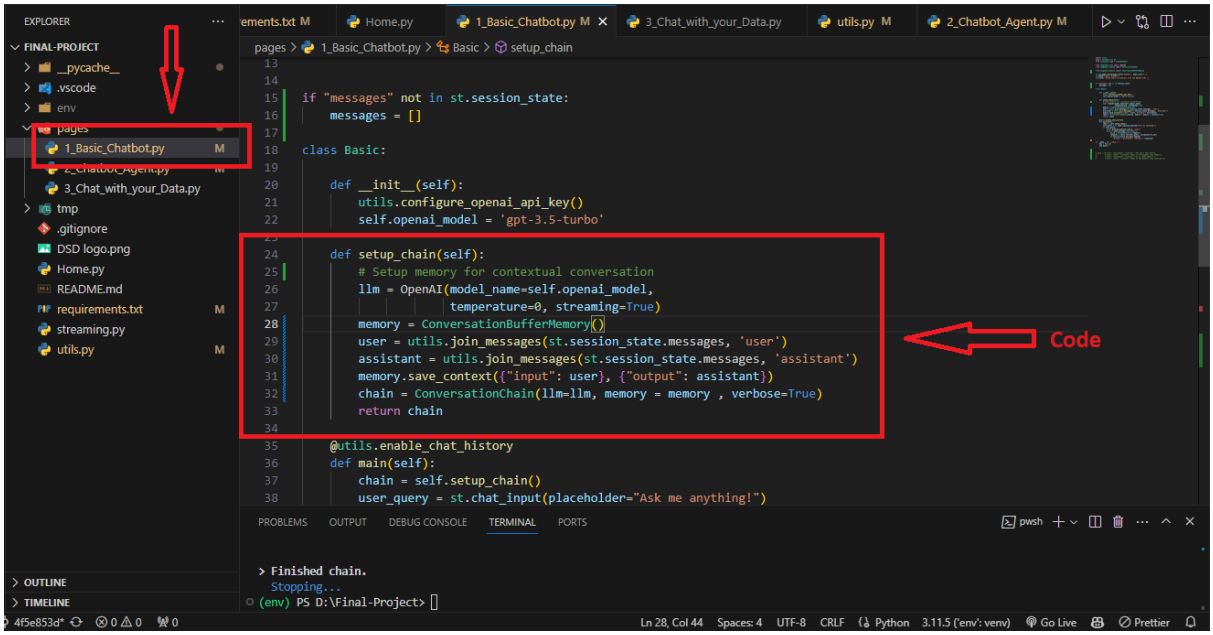
9. Now, add the following code in the same file before the **Basic Class** constructor begins. You can **copy** the **code** from below.

```
if "messages" not in st.session_state:
    st.session_state["messages"] = []
```



10. Now, change the code of **setup_chain** function in the same file with the following code. You can **copy** the **code** from below.

```
    def setup_chain(self):
        # Setup memory for contextual conversation
        llm = OpenAI(model_name=self.openai_model,
                     temperature=0, streaming=True)
        memory = ConversationBufferMemory()
        user = utils.join_messages(st.session_state.messages,
'user')
        assistant = utils.join_messages(st.session_state.messages,
'assistant')
        memory.save_context({"input": user}, {"output": assistant})
        chain = ConversationChain(llm=llm, memory = memory ,
verbose=True)
        return chain
```



11. Now, run the code. Open **Terminal** and write **streamlit run ./Home.py**