NoteBot - Code Explanation

```
import streamlit as st
```

Imports Streamlit library to build the web application UI.

```
from PyPDF2 import PdfReader
```

Imports PdfReader class from PyPDF2 to read and extract text from PDF files.

```
from langchain.chains.combine_documents import create_stuff_documents_chain
```

Imports a function that allows combining document chunks with a language model for answering questions.

```
from langchain_text_splitters import RecursiveCharacterTextSplitter
```

Imports a utility to split long text into smaller chunks for better embedding and retrieval.

```
from langchain_openai import OpenAIEmbeddings, ChatOpenAI
```

Imports OpenAIEmbeddings to convert text into vectors and ChatOpenAI to access OpenAI's chat-based LLM.

```
from langchain_community.vectorstores import FAISS
```

Imports FAISS vector database to store and search embeddings efficiently.

```
from langchain.prompts import ChatPromptTemplate
```

Imports a template system for customizing prompts for the LLM.

```
OpenAI_API_KEY="Paste your OpenAI API key here"
```

Defines the API key for connecting to OpenAI (user should replace with their actual key).

```
st.header("NoteBot")
```

Displays the app's main header on the Streamlit interface.

```
with st.sidebar:
```

Creates a sidebar section in the Streamlit app.

```
st.title("My Notes")
```

Sets the title of the sidebar.

```
file = st.file_uploader("Upload notes PDF and start asking questions", type="pdf")
```

Provides a file uploader widget in the sidebar that accepts PDF files.

```
if file is not None:
```

Checks if a PDF file is uploaded.

```
my_pdf=PdfReader(file)
```

Loads the uploaded PDF file into PdfReader object.

```
text=""
```

Initializes an empty string to store extracted text.

```
for page in my_pdf.pages:
```

Loops through each page of the PDF.

```
text += page.extract_text()
```

Extracts and concatenates text from each page into a single string.

```
splitter = RecursiveCharacterTextSplitter(chunk_size=300, chunk_overlap=50,
length_function=len)
```

Initializes text splitter with chunk size of 300 characters and 50-character overlap.

```
chunks=splitter.split_text(text)
```

Splits the extracted text into smaller chunks.

```
embeddings=OpenAIEmbeddings(api_key=OpenAI_API_KEY)
```

Creates an OpenAI embeddings object to transform chunks into vector embeddings.

```
vector_store=FAISS.from_texts(chunks,embeddings)
```

Stores the embeddings in a FAISS vector store for semantic search.

```
user_query = st.text_input("Type your query here")
```

Adds a text input field for the user to ask questions.

```
if user_query:
```

Checks if the user has entered a query.

```
matching_chunks=vector_store.similarity_search(user_query)
```

Performs semantic search on the stored embeddings to find chunks relevant to the query.

```
llm = ChatOpenAI(api_key=OpenAI_API_KEY, max_tokens=300, temperature=0,
model="gpt-3.5-turbo")
```

Initializes the LLM with API key, token limit, deterministic responses (temperature=0), and model specification.

```
customized_prompt = ChatPromptTemplate.from_template(...)
```

Defines a custom prompt template instructing the LLM to act as a tutor and handle unknowns gracefully.

```
chain = create_stuff_documents_chain(llm, customized_prompt)
```

Creates a chain that combines the documents and prompt with the LLM for answering queries.

```
output=chain.invoke({"input":user_query, "input_documents": matching_chunks})
```

Executes the chain with the user query and the matched document chunks to generate an answer.

```
st.write(output)
```

Displays the final answer to the user in the Streamlit app.