1. Importing Necessary Libraries
python

```
from langchain.prompts import ChatPromptTemplate
from langchain_ollama.llms import OllamaLLM
import streamlit as st
```

langchain.prompts: Imports ChatPromptTemplate from LangChain. This is used for creating structured prompts that guide the behavior of the language model.
langchain_ollama.llms: Imports OllamaLLM from the langchain_ollama module. This class allows you to interact with the Ollama models (specifically, Deepseek-R1 in this case).
streamlit as st: Imports the Streamlit library as st. Streamlit is used to create the interactive web application that lets users input questions and get responses from the model.
2. Setting the Title of the Streamlit App
python

```
st.title("My ChatBot using Deepkseek-R1")
```

This sets the title of the web page when it's displayed in the browser. In this case, the title is "My ChatBot using Deepseek-R1", indicating the purpose of the app.
3. Defining the Prompt Template
python

```
template = """question: {question}
Answer = Generate the answer step by step"""
```

template: This defines the structure of the prompt that will be fed into the model. The {question} is a placeholder, which will be replaced with the actual user's input later. The template asks the model to answer the question "step by step", which influences the way the model provides answers (breaking it down into parts or explaining each step clearly).
4. Creating a Chat Prompt Template
python

```
prompt = ChatPromptTemplate.from_template(template)
```

ChatPromptTemplate.from_template(template): This creates a ChatPromptTemplate object using the defined template. The from_template method allows the template to be reused and formatted with user inputs dynamically (i.e., plugging in the user's question into the {question} placeholder in the template).
5. Creating the Model Instance
python

```
model = OllamaLLM(model = "deepseek-r1")
```

OllamaLLM(model="deepseek-r1"): This line creates an instance of the OllamaLLM class, which is responsible for interacting with the Deepseek-R1 model. The model="deepseek-r1" argument specifies that the Deepseek-R1 model will be used to generate answers.
6. Chaining the Prompt and the Model
python

```
chain = prompt | model
```
chain = prompt | model: This creates a chain of operations. The | operator is used to combine the prompt and the model. The prompt is first formatted with the user's input, and then the formatted prompt is passed to the model for generating a response.
This chain object can be used to invoke the model and get responses based on the question asked by the user.
7. User Input Field in Streamlit
python

```
question = st.text_input("Enter you question here:")
```
st.text_input("Enter your question here:"): This creates an interactive input box on the web page, where users can type in their question. The text entered by the user is stored in the question variable.
8. Condition to Check If User Entered a Question
python

```
if question:
```
This checks if the user has entered a question (i.e., if question is not an empty string).
9. Handling the Response Generation with Try-Except
python

```
try:
    formatted_prompt = prompt.format(question=question)
    response = chain.invoke({"question": question})
    st.write(response)
except Exception as e:
    st.write(f"Error: {e}")
```
formatted_prompt = prompt.format(question=question): This takes the prompt template and formats it with the user's question. The placeholder {question} in the template is replaced with the actual question entered by the user.

response = chain.invoke({"question": question}): The chain object, which combines the prompt and model, is invoked with the formatted question. This sends the formatted question to the Deepseek-R1 model, which processes it and generates a response. The response is stored in the response variable.

st.write(response): Once the response is generated, Streamlit displays it on the web page so the user can see the answer.

except Exception as e:: If any error occurs during the process (such as issues with the model or formatting), the code will catch the exception.

st.write(f"Error: {e}"): If there’s an error, an error message is displayed to the user on the Streamlit web page.

Full Flow of Code Execution:
The user visits the web app and sees a prompt that says: "Enter your question here".

The user types a question into the input field.
The app formats the question into a structured prompt (using ChatPromptTemplate) and sends it to the Deepseek-R1 model.
The model processes the question and generates a response, which is displayed on the app.
If any error occurs during the process (e.g., a problem with formatting or invoking the model), the error message is displayed instead.
In Summary:
LangChain is used to create and format a prompt to guide the model's response.
Streamlit provides an interactive interface for users to input their questions.
The Deepseek-R1 model is used to generate step-by-step answers to the user’s questions.
Error handling ensures that the user is informed of any issues in the process.
This project demonstrates the integration of NLP models with web applications, using LangChain for prompt management and Streamlit for creating user-friendly interfaces.