Chat Prompt Helper with LangChain This project demonstrates a simple yet effective way to use LangChain to create a conversational helper. The code sets up a prompt template with pre-defined roles and messages, then invokes a large language model to generate a response based on a specific user query and an error message.

Description The script initializes a chat prompt template designed to mimic a conversation between an experienced developer and a user preparing for a Data Structures and Algorithms (DSA) interview. The system message establishes the persona of an expert developer, while the human message provides the context of the user's problem, including a specific error they are facing.

The code then uses this template to invoke a large language model, providing the language ("python") and the error ("; missing error") as variables. The final output is the model's tailored response to this specific problem.

Prerequisites Before running this code, you need to have the following libraries installed:

LangChain: The framework for building LLM applications.

A compatible model library: This example assumes you have a model library like google-generativeai, openai, or another one integrated with LangChain.

Installation To install the necessary packages, you can use pip:

pip install langchain langchain-google-genai

Note: The second package, langchain-google-genai, is a placeholder. Please replace it with the specific package for your model.

Code Here is a full breakdown of the provided code:

from langchain_core.prompts import ChatPromptTemplate from langchain_google_genai import ChatGoogleGenerativeAI

The prompt template defines the structure and roles of the conversation.

It includes a ‘system’ role to set the persona and a ‘human’ role for the user’s input.

Variables like `{language}` and `{error}` are placeholders filled later.

```
template_message = [ (“system”, “you are an experianced {language}
developer”), (“human”, “i am preparing for DSA interview and practising
question. help me with the error{error}”)]
```

Create the ChatPromptTemplate object from the list of messages.

```
Prompt_template_with_message =
ChatPromptTemplate.from_messages(template_message)
```

Invoke the template with specific values for the placeholders.

This creates a PromptValue object with the full, formatted prompt.

```
Prompt_input = Prompt_template_with_message.invoke({“language”:
“python”, “error”: “; missing error”}) print(Prompt_input)
```

Initialize your specific model. Replace this with your actual model setup.

model = ChatGoogleGenerativeAI(model="gemini-1.5-flash")

Invoke the model with the prepared prompt to get the generated response.

response = model.invoke(Prompt_input) print(response.content)

How to Use Install the prerequisites: Follow the installation steps above.

Add your API key: Ensure your environment is configured with the necessary API key for the model you are using. For example, for Google's Gemini models, you would need to set the GOOGLE_API_KEY environment variable.

Run the script: Simply execute the Python file.

Customization You can easily adapt this code to different use cases by modifying the invoke call:

Change the developer's language: Modify the language variable, e.g., {"language": "javascript", ...}.

Provide a different error: Replace "; missing error" with any specific error message, e.g., {"error": "TypeError: 'str' object is not callable"}.

Change the entire persona: Edit the template_message list to change the system and human messages to fit a new purpose, such as a creative writing assistant or a technical support bot.