Rule-Based Chatbot Development

The goal of this project was to develop a rule-based chatbot using Python and HTML. The chatbot uses predefined pattern-response pairs with regular expressions for pattern matching and dynamic responses, making the conversation more interactive. This document describes our approach, the techniques we used, the challenges faced during the development process, and how we addressed them.

**SimpleBot Class Design:**

Our SimpleBot class was designed to handle user input and respond appropriately. During it’s development, the following steps were taken to design it:

* **Pattern-Response Pairs**: We defined 15 regular expressions using regex to capture specific user input patterns. For each regex, we paired a corresponding response, some of which were dynamic (inserting user-provided text into the response).

We also designed a pattern matching that creates a more realistic conversation by making the chatbot use a regular expression to identify user intent. We used capturing groups parentheses in regex to extract certain parts of the user's message, then formatted the responses using Python’s .format() method to insert these captured words into the response.

For example:

* r"do you (like|love) (.\*)": That pattern captures whether the user asks if the bot likes or loves something and inserts the subject (like example: pizza) into the response.
* Response: "I don’t have feelings, but I imagine pizza is pretty interesting! Do you like pizza?"

However,creating these patterns was not as easy as it seems because we encountered some challenges designing regex patterns that were flexible enough to capture various sentence structures without over-matching or misinterpreting user input. For instance, while working with the “like/love” pattern, we initially captured too much of the user input, which led to generating incorrect responses like "I imagine like is pretty interesting!", later we were able to resolve this issue by adjusting the regex to correctly isolate and differentiate the subject from the verb.

* **Generic Responses**: We Also created 5 generic responses in case the user's input did not match any pattern. Those responses were designed to maintain the conversation flow so that even if the user’s input does not match, the conversation will continue.

**REST API**

In our project, we have a GET request which is used to fetch the webpage, such as the home, About and Documentation page. The POST request is used to send a response to the chatbot handler which will interpret the response and send back a message to the user. For my part, I did the chatbot handler class, the purpose of this was meant as a way to integrate the SimpleBot class. The way it does this is that, the post request will extract the message from the user, and will determine if it’s a valid or invalid response. If its invalid it will go to the default error message we integrated in hopes that it can have the user rewrite their question to something the SimpleBot could understand while keeping the conversation going. If its valid it will choose an appropriate response from the SimpleBot class and send it back to the user. Essentially, this class is just a way for the SimpleBot class and the user to communicate with each other.

**UI webpage**

We implemented Javascript scripting to render the users input to display in a element container along with the chatbots reponse. Also created a nav bar which explores the documentation and About page for further information on the project. Moreover for consistency the design is simple and the chat window is easy to read and use. For the webpage we decided to use a lightweight library called Flask which is known for creating quick web frameworks and is easy to organize template files and CSS styling.