CREATE A CHATBOT IN PYTHON

GROUP-5 ARTIFICAL INTELLIGENCE

TEAM MEMBER

950921104004: ANUSHIYA.K

Phase-1 Document Submission

Project: create a chatbot in python

OBJECTIVE:

The challenge is to create a chatbot in python that provides exceptional customer service, answering user queries on a website or application. The objective is to deliver high-quality support to users, ensuring a positive user experience and customer and customer satisfaction.

1.FUNCTIONALLITY:

Define the scope of the chatbot's abilities, including answering common questions, providing guidance, and directing users to appropriate resources.

- **Answering Questions:** Many chatbots are designed to provide information and answer questions based on their knowledge or access to databases. For example, a customer support chatbot might help users troubleshoot issues with a product or service.
- **Task Automation:** Chatbots can automate tasks, such as setting reminders, making appointments, sending notifications, or even ordering products online. These tasks are often performed by interfacing with external systems or APIs.
- **Natural Language Processing (NLP):** A key aspect of chatbot functionality is its ability to understand and generate natural language text. NLP techniques are used to process and interpret user input and to generate human-like responses.
- **Conversation Management:** Chatbots need to manage the flow of a conversation, including handling multiple turns, context switching, and maintaining a coherent dialogue. This involves tracking user intent and responding appropriately.

PYTHON CODDING:

```
import random


# Define a list of common questions and their corresponding answers
common_questions = {
    "What's the weather like today?": "I'm sorry, I can't provide real-time weather information.",
    "Tell me a joke": "Why did the chicken cross the road? To get to the other side!",
```

```
    "Who won the World Series in 2020?": "The Los Angeles Dodgers won the World Series in 2020.",
    "How does photosynthesis work?": "Photosynthesis is the process by which plants convert sunlight into energy...",
    # Add more questions and answers as needed
}

# Function to handle user queries
def chatbot_response(user_input):
    if user_input in common_questions:
        return common_questions[user_input]
    else:
        return "I'm sorry, I don't know the answer to that question."

# Main loop for the chatbot
while True:
    user_input = input("You: ")
    if user_input.lower() == 'exit':
        print("Chatbot: Goodbye!")
        break
    response = chatbot_response(user_input)
   print("Chatbot:", response)
```

2.USER INTERFACE:

Determine where the chatbot will be integrated (website, app) and design a user-friendly interface for interactions.

- **Text Input:** Users typically interact with chatbots by typing text into an input field. This text input serves as the user's way of expressing their questions, requests, or commands.
- **Chat Window:** The chat window is where the conversation between the user and the chatbot takes place. It displays the user's messages and the chatbot's responses in a conversational format, often resembling a chat conversation.
- **Send Button:** A "Send" or "Submit" button allows users to submit their input to the chatbot. Clicking this button sends the user's message to the chatbot for processing.

- **Voice Input:** Some chatbots support voice input, allowing users to speak to the bot instead of typing. This is common in voice-activated chatbots and virtual assistants.
- **Rich Media Support:** Depending on the chatbot's capabilities, it may support sending or receiving images, videos, documents, and other types of rich media as part of the conversation.
- **Quick Replies/Buttons:** To guide users or offer predefined options, chatbots can provide quick reply buttons or options that users can click instead of typing out responses. This can simplify interactions for users.
- **Emojis and Emoticons:** Adding emojis or emoticons can make the conversation more engaging and expressive, helping to convey emotions or reactions.
- **Typing Indicators:** A typing indicator informs users when the chatbot is processing their request. It's a visual cue that indicates the bot is "thinking" or composing a response.
- **User Feedback Mechanisms:** Chatbots may include mechanisms for users to provide feedback, such as thumbs-up/down buttons, ratings, or comments on the quality of service.
- **User Instructions:** Clear and concise instructions or hints can help users understand how to interact with the chatbot effectively. For example, instructing users to type "help" for assistance.

PYTHON CODDING:

```
from flask import Flask, render_template, request, jsonify
import spacy

app = Flask(_name_)

# Load the English language model
nlp = spacy.load("en_core_web_sm")

# Function to handle user queries with NLP
def chatbot_response(user_input):
    # Process user input with spaCy
    doc = nlp(user_input)

    # Extract named entities (if any)
    named_entities = [ent.text for ent in doc.ents]

    # Determine the main verb in the user's input
    main_verb = None
    for token in doc:
        if "VB" in token.pos_:
            main_verb = token.text
            break
```

```
    # You can implement more advanced logic based on the processed user input
    response = "I'm sorry, I don't understand your question."

    if named_entities:
        response = f"I detected named entities: {', '.join(named_entities)}."

    if main_verb:
        response = f"Your main verb is: {main_verb}."

    return response

# ... rest of the Flask app code (routes and HTML template) remains the same ...

if _name_ == '_main_':
    app.run(debug=True)
```

3.NATURAL LANGUAGE PROCESSING (NLP):

Implement NLP techniques to understand and process user input in a conversational manner.

- **Text Understanding:** NLP enables chatbots to understand and interpret the text input provided by users. This includes tasks like:
- **Tokenization:** Breaking down input text into individual words or tokens.
- **Part-of-Speech Tagging:** Identifying the grammatical parts of speech (e.g., nouns, verbs, adjectives) for each word in a sentence.
- **Intent Recognition:** One of the most critical aspects of NLP in chatbots is recognizing the user's intent or the purpose behind their message. This involves classifying user queries into specific categories or actions that the chatbot can respond to. For example, identifying whether a user is asking for weather information, making a reservation, or seeking product recommendations.
- **Contextual Understanding:** NLP allows chatbots to maintain context throughout a conversation. This means the chatbot can remember previous messages and use that information to provide more relevant and coherent responses. Contextual understanding helps avoid misunderstandings and enables more natural conversations.
- **Language Generation:** In addition to understanding user input, NLP also enables chatbots to generate human-like responses. This involves:
- **Text Generation:** Creating text responses that are grammatically correct and contextually appropriate.
- **Language Style:** Adapting the tone and style of responses to match the user's preferences or the chatbot's persona.
- **Multilingual Support:** NLP can enable chatbots to communicate in multiple languages, expanding their reach to a broader user base.

- **Dialog Management:** NLP helps chatbots manage the flow of a conversation. It allows the chatbot to decide when to ask for clarification, offer suggestions, or move to the next step in a multi-turn interaction.

PYTHON PROGRAM:

```
def clean_text(text):
    text=re.sub('-',' ',text.lower())
    text=re.sub('[.]',' . ',text)
    text=re.sub('[1]',' 1 ',text)
    text=re.sub('[2]',' 2 ',text)
    text=re.sub('[3]',' 3 ',text)
    text=re.sub('[4]',' 4 ',text)
    text=re.sub('[5]',' 5 ',text)
    text=re.sub('[6]',' 6 ',text)
    text=re.sub('[7]',' 7 ',text)
    text=re.sub('[8]',' 8 ',text)
    text=re.sub('[9]',' 9 ',text)
    text=re.sub('[0]',' 0 ',text)
    text=re.sub('[,]',' , ',text)
    text=re.sub('[?]',' ? ',text)
    text=re.sub('[!]',' ! ',text)
    text=re.sub('[$]',' $ ',text)
    text=re.sub('[&]',' & ',text)
    text=re.sub('[/]',' / ',text)
    text=re.sub('[:]',' : ',text)
    text=re.sub('[;]',' ; ',text)
    text=re.sub('[*]',' * ',text)
    text=re.sub('[\']',' \' ',text)
    text=re.sub('[\"]',' \" ',text)
    text=re.sub('\t',' ',text)
```

```
    return text
from flask import Flask, render_template, request, jsonify
import spacy

app = Flask(_name_)

# Load the English language model
nlp = spacy.load("en_core_web_sm")

# Function to handle user queries with NLP
def chatbot_response(user_inpu):
    # Process user input with spaCy
    doc = nlp(user_input)

    # Extract named entities (if any)
    named_entities = [ent.text for ent in doc.ents]

    # Determine the main verb in the user's input
    main_verb = None
    for token in doc:
        if "VB" in token.pos_:
            main_verb = token.text
            break

    # You can implement more advanced logic based on the processed user input
    response = "I'm sorry, I don't understand your question."

    if named_entities:
        response = f"I detected named entities: {', '.join(named_entities)}."
```

```
    if main_verb:
        response = f"Your main verb is: {main_verb}."

    return response

# ... rest of the Flask app code (routes and HTML template) remains the same ...

if _name_ == '_main_':
    app.run(debug=True)
```

OUTPUT:

encoder_inputs	decoder_targets	decoder_inputs	
0	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>
1	i ' m fine . how about yourself ?	i ' m pretty good . thanks for asking . <end>	<start> i ' m pretty good . thanks for asking...
2	i ' m pretty good . thanks for asking .	no problem . so how have you been ? <end>	<start> no problem . so how have you been ? ...
3	no problem . so how have you been ?	i ' ve been great . what about you ? <end>	<start> i ' ve been great . what about you ? ...
4	i ' ve been great . what about you ?	i ' ve been good . i ' m in school right now ...	<start> i ' ve been good . i ' m in school ri...
5	i ' ve been good . i ' m in school right now .	what school do you go to ? <end>	<start> what school do you go to ? <end>
6	what school do you go to ?	i go to pcc . <end>	<start> i go to pcc . <end>
7	i go to pcc .	do you like it there ? <end>	<start> do you like it there ? <end>
8	do you like it there ?	it ' s okay . it ' s a really big campus . <...	<start> it ' s okay . it ' s a really big cam...
9	it ' s okay . it ' s a really big campus .	good luck with school . <end>	<start> good luck with school . <end>

4.RESPONSES:

Plan responses that the chatbot will offer, such as accurate answers, suggestions, and assistance.

- **Content:** The content of a response can vary widely depending on the chatbot's purpose and capabilities. It may include text, links, images, videos, documents, or other types of information or media that are relevant to the user's query.
- **Contextual Understanding:** Effective chatbots aim to understand the context of the conversation to provide relevant responses. This involves considering previous user messages and maintaining context throughout the conversation.
- **Natural Language:** Responses are typically generated in natural language, making them easy for users to understand and engage with. Natural language processing (NLP) techniques are often used to generate coherent and contextually relevant responses.
- **Personalization:** Some chatbots can personalize their responses based on user data, preferences, or history. Personalization enhances the user experience by tailoring responses to individual users.
- **Multimodal Responses:** Depending on the platform and capabilities, chatbots may support multimodal responses, such as combining text with images, cards, or buttons to provide a more interactive experience.
- **Dynamic Responses:** Chatbots can generate dynamic responses that change based on user input or other factors. For example, a weather chatbot might provide current weather conditions that vary by location.
- **Error Handling:** Chatbots should be equipped to handle errors or invalid user input by providing informative error messages or offering suggestions to guide the user toward a valid query.
- **Variety of Responses:** To make interactions more engaging, chatbots may offer a variety of responses, including informative responses, clarifying questions, humor, greetings, or acknowledgments.
- **Response Time:** A responsive chatbot provides timely responses to maintain the flow of the conversation. Long response times can lead to user frustration.
- **Fallback Responses:** Fallback responses are default messages that the chatbot uses when it doesn't understand the user's query or cannot perform a specific task. They are designed to keep the conversation going or direct the user to other resources

PYTHON CODDING:

```
from flask import Flask, render_template, request, jsonify

import spacy


app = Flask(_name_)

# Load the English language model

nlp = spacy.load("en_core_web_sm")
```

```
# Define a dictionary of predefined responses
predefined_responses = {
    "tell_me_a_joke": "Why did the scarecrow win an award? Because he was outstanding in his field!",
    "greet": "Hello! How can I assist you today?",
    "goodbye": "Goodbye! If you have more questions, feel free to return.",
}

# Function to handle user queries with NLP
def chatbot_response(user_input):
    # Process user input with spaCy
    doc = nlp(user_input)

    # Extract named entities (if any)
    named_entities = [ent.text for ent in doc.ents]

    # Determine the main verb in the user's input
    main_verb = None
    for token in doc:
        if "VB" in token.pos_:
            main_verb = token.text
            break

    # Implement logic to select a response based on user input
    if "tell me a joke" in user_input.lower():
        return predefined_responses["tell_me_a_joke"]
    elif any(entity in ["weather", "forecast"] for entity in named_entities):
        return "I'm sorry, I can't provide real-time weather information."
```

```
    elif "greet" in user_input.lower():
        return predefined_responses["greet"]
    elif "goodbye" in user_input.lower():
        return predefined_responses["goodbye"]
    else:
        return "I'm sorry, I don't understand your question."

# ... rest of the Flask app code (routes and HTML template) remains the same ...

if _name_ == '_main_':
    app.run(debug=True)
```

5.INTEGRATION:

Decide how the chatbot will be integrated with the website or app.

- **Data Integration:** Chatbots often need to access and retrieve data from external sources to provide users with accurate and up-to-date information. Data integration involves connecting the chatbot to databases, APIs, or data storage systems to fetch relevant data.
- **API Integration:** Many chatbots integrate with third-party APIs (Application Programming Interfaces) to access external services or perform actions. For example, an e-commerce chatbot might integrate with a payment gateway API for processing transactions.
- **Platform Integration:** Chatbots can be deployed on various platforms, including websites, messaging apps (e.g., Facebook Messenger, Slack), and voice assistants (e.g., Amazon Alexa, Google Assistant). Integration ensures that the chatbot functions seamlessly within the chosen platform.
- **CRM Integration:** In customer service applications, chatbots often integrate with Customer Relationship Management (CRM) systems to access customer data, history, and preferences, enabling more personalized interactions.
- **Authentication and Authorization:** Integration often requires secure authentication and authorization mechanisms to ensure that the chatbot has the appropriate permissions to access external systems or perform specific actions.
- **Real-time Communication:** Chatbots may need to integrate with real-time communication tools such as chat applications or notification services to deliver timely updates or alerts to users.
- **Natural Language Processing (NLP) Services:** Chatbots often integrate with NLP services or libraries to improve their understanding of user queries and generate more contextually relevant responses.
- **Payment Gateways:** E-commerce chatbots may integrate with payment gateways to facilitate transactions securely within the chat interface.
- **Bot-to-Bot Integration:** In some cases, chatbots may need to communicate with other chatbots or virtual assistants to collaborate on tasks or share information.

PYTHON CODDING:

```
# chatbot_server.py
from flask import Flask, request, jsonify

app = Flask(_name_)

# Your chatbot logic here (e.g., using predefined_responses as before)
def chatbot_response(user_input):
    # Implement your chatbot logic here
    return "Chatbot response for: " + user_input

@app.route('/chat', methods=['POST'])
def chat():
    user_input = request.json.get('user_input')
    response = chatbot_response(user_input)
    return jsonify({'response': response})

if _name_ == '_main_':
    app.run(debug=True)
```

```
conda create --name chatbot-env python=3.8
conda activate chatbot-env
conda install flask
```

```
python chatbot_server.py
```

6.TESTING AND IMPROVEMENT:

Continuously test and refine the chatbot's performance based on user interactions.

Testing:

- **Functional Testing:** This involves verifying that the chatbot performs its intended functions correctly. Test cases are designed to cover various scenarios, including user queries, input validation, and task execution. Functional testing helps identify and rectify bugs or errors in the chatbot's functionality.

- **Usability Testing:** Usability testing assesses how easy it is for users to interact with the chatbot. Testers, often representing the target audience, provide feedback on the chatbot's user interface, language understanding, and overall user experience.
- **Performance Testing:** Performance testing evaluates the chatbot's response time and scalability. It helps determine whether the chatbot can handle a high volume of simultaneous users without slowing down or crashing.
- **Integration Testing:** When a chatbot is integrated with external systems or APIs, integration testing ensures that these connections work as expected. It verifies that data is exchanged correctly, and actions are performed without issues.
- **Security Testing:** Security testing identifies vulnerabilities in the chatbot, such as input validation, authentication, and data protection issues. Ensuring the chatbot handles user data securely is essential, especially if it deals with sensitive information.
- **Regression Testing:** As the chatbot evolves and new features are added, regression testing verifies that existing functionality still works correctly after each update. It prevents the introduction of new bugs.

Improvement:

- **User Feedback Analysis:** Collect and analyze user feedback to identify areas where the chatbot can be improved. Feedback can come from user surveys, direct user interactions, or monitoring chatbot logs.
- **Performance Optimization:** Based on performance testing results, optimize the chatbot's response time and scalability. This might involve optimizing code, enhancing server infrastructure, or using caching mechanisms.
- **Natural Language Processing (NLP) Enhancement:** Continuously improve the chatbot's NLP capabilities by fine-tuning language models, training data, and entity recognition to better understand user queries.
- **Content Updates:** Keep the chatbot's content up-to-date by regularly adding new information, responses, or options based on changing user needs or updated data sources.
- **Error Handling:** Enhance the chatbot's ability to handle errors gracefully. Provide informative error messages and recovery options for users when the chatbot encounters issues or misunderstands queries.
- **Personalization:** Implement personalization features to make the chatbot's responses more tailored to individual users based on their preferences, history, and behavior.
- **A/B Testing:** Conduct A/B testing to compare different versions of the chatbot, such as variations in responses or user interface design, to determine which performs better in terms of user engagement and satisfaction.
- **Continuous Learning:** Implement mechanisms for the chatbot to learn from user interactions and improve its responses over time. Machine learning and reinforcement learning techniques can be used for this purpose.

PYTHON CODDING:

Testing and improvement

```
# chatbot_server.py
```

```
# Import necessary libraries and modules
import json
from flask import Flask, request, jsonify

app = Flask(_name_)

# Initialize an empty list to store chat logs
chat_logs = []

@app.route('/chat', methods=['POST'])
def chat():
    user_input = request.json.get('user_input')
    response = chatbot_response(user_input)

    # Log the conversation
    chat_logs.append({'user': user_input, 'chatbot': response})

    return jsonify({'response': response})

@app.route('/feedback', methods=['POST'])
def feedback():
    data = request.get_json()
    user_feedback = data.get('feedback')

    # Process and store user feedback
    # You can use this feedback to make improvements to the chatbot

    return jsonify({'message': 'Thank you for your feedback!'})
```

```
if _name_ == '_main_':
    app.run(debug=True0)
```

OUTPUT:

	question	answer
0	hi, how are you doing?	i'm fine. how about yourself?
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.
2	i'm pretty good. thanks for asking.	no problem. so how have you been?
3	no problem. so how have you been?	i've been great. what about you?
4	i've been great. what about you?	i've been good. i'm in school right now.