CREATE A CHATBOT IN PYTHON

GROUP:5 ARTIFICIAL INTELLIGENCE

TEAM MEMBER

950921104029 : RESHMA.R

Phase-1 Document Submission

Project: creating chatbot in python

OBJECTIVE:

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

FUNCTIONALITY:

Defining the scope of a chatbot's functionality in Python can be achieved by creating a set of functions or rules that specify how the chatbot should respond to different inputs. Below is a basic example of a chatbot's functionality to answer common questions,provide guidance and direct users to appropriate resources

1.We define a dictionary called `common_questions` that maps common user questions to their corresponding answers.

2.we create a `chatbot_response` function that takes user input as an argument, 3. converts it to lowercase to make it case-insensitive, and checks if it matches any of the common questions. If it does, the function returns the corresponding answer; otherwise, it provides a generic response.

3.We use the `input` function to get user input and then call the `chatbot response` function to generate a response.

4.Finally, we print the chatbot's response to the user.

PYTHON PROGRAM:

```
import tensorflow as tf

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from tensorflow.keras.layers import TextVectorization
```

```
import re,string
from tensorflow.keras.layers import LSTM,Dense,Embedding,Dropout,LayerNormalization
import random
# Define a list of common questions and their corresponding answers
common_questions =
{
    "What is your name?": "I am a chatbot.",
    "How are you?": "I'm just a computer program, so I don't have feelings, but thanks for asking!",
    "What can you do?": "I can answer questions, provide guidance, and direct you to resources.",
}


# Define a function to handle user input
def chatbot_response(user_input):
    user_input = user_input.lower()  # Convert user input to lowercase for case-insensitivity


    # Check if the user input is a common question
    if user_input in common_questions:
        return common_questions[user_input]
    else:
        # If the input is not a common question, provide a generic response
        return "I'm not sure how to respond to that. Please ask me a different question."


# Example usage:
user_input = input("You: ")  # Get user input
bot_response = chatbot_response(user_input)
print("Chatbot:", bot_response)
```

USER INTERFACE:

A user interface (UI) in the context of a chatbot refers to the means through which a user interacts with the chatbot. It is the interface that facilitates communication between the user and the chatbot. The design of the user interface significantly impacts the user experience and the

effectiveness of the chatbot. Here are some key aspects to consider when discussing the user interface in a chatbot:

Creating a user-friendly interface for a chatbot depends on the platform where you want to integrate it. Here, I'll show you an example of how to create a simple command-line interface (CLI) for interactions using Python. You can adapt this code to integrate the chatbot into a website, app, or any other platform of your choice.

- **Medium of Interaction:** The user interface can take various forms depending on where the chatbot is deployed. Some common mediums include:
- **Text-Based:** This is the most common form, where users type text messages or question to the chatbot and receive text-based responses.
- **Voice-Based:** In voice-based chatbots, users communicate with the bot through spoken language. Platforms like Amazon Alexa and Google Assistant use voice-based UIs.
- **Graphical User Interface (GUI):** In some cases, chatbots are integrated into graphical interfaces, such as mobile apps or web applications, where users can interact with the chatbot through buttons, menus, and visual elements.
- **Conversational Flow:** A chatbot's user interface should be designed to facilitate natural and intuitive conversations. This includes handling multi-turn conversations and context switching when the user changes topics.
- **Feedback and Prompts:** Providing clear feedback to the user is essential for a good user interface. The chatbot should acknowledge user input, confirm actions, and provide helpful prompts to guide the conversation.
- **Personalization:** User interfaces can be personalized to enhance the user experience. This might include addressing the user by name, remembering past interactions, and adapting responses based on user preferences or history.
- **Error Handling:** The UI should be designed to handle user errors and misunderstandings gracefully. It should provide helpful error messages and suggest possible corrections.

PYTHON PROGRAM:

```
import random

df=pd.read_csv('/kaggle/input/simple-dialogs-for-
chatbot/dialogs.txt',sep='\t',names=['question','answer'])

print(f'Dataframe size: {len(df)}')

df.head()

# Define a dictionary of common questions and answers

common_questions = {
```

```
    "What is your name?": "I am a chatbot.",
    "How are you?": "I'm just a computer program, so I don't have feelings, but thanks for asking!",
    "What can you do?": "I can answer questions, provide guidance, and direct you to resources.",
}

# Function to handle user input
def chatbot_response(user_input):
    user_input = user_input.lower()  # Convert user input to lowercase for case-insensitivity

    # Check if the user input is a common question
    if user_input in common_questions:
        return common_questions[user_input]
    else:
        # If the input is not a common question, provide a generic response
        return "I'm not sure how to respond to that. Please ask me a different question."

# Function to display the chatbot interface
def chat_interface():
    print("Chatbot: Hello! How can I assist you today? (Type 'exit' to end the conversation)")

    while True:
        user_input = input("You: ")

        if user_input.lower() == 'exit':
```

```
            print("Chatbot: Goodbye!")
            break


        bot_response = chatbot_response(user_input)
        print("Chatbot:", bot_response)


# Run the chatbot interface
if __name__ == "__main__":
    chat_interface()
def print_conversation(texts):
    for text in texts:
        print(f'You: {text}')
        print(f'Bot: {chatbot(text)}')
        print('========================')
```

NATURAL LANGUAGE PROCESSING:

Natural Language Processing, is a fundamental component of chatbots and plays a crucial role in their ability to understand and generate human language. NLP in chatbots refers to the set of techniques and technologies used to enable the chatbot to interact with users in a natural, human-like way by processing and understanding text or speech input. Here's how NLP is applied in chatbots:

- **Text Understanding:** Chatbots use NLP to understand and interpret the text-based input provided by users. This includes tasks such as tokenization (breaking text into words or tokens), part-of-speech tagging (identifying the grammatical roles of words), and syntactic parsing (analyzing the sentence structure).

- **Intent Recognition:** NLP helps chatbots recognize the intent behind user input. It involves identifying what the user wants or the action they are trying to perform. For example, if a user asks, "What's the weather like today?" the chatbot should recognize the intent as a weather inquiry.
- **Entity Recognition:** NLP allows chatbots to identify and extract specific pieces of information or entities from user input. For instance, in the question

"What's the weather like in New York today?" the entity "New York" should be recognized as a location.

- **Context Management:** Chatbots use NLP to manage conversational context. This involves remembering previous interactions and maintaining context throughout the conversation. Context helps the chatbot provide relevant responses and answer follow-up questions accurately.
- **Sentiment Analysis:** NLP can be applied to analyze the sentiment or emotional tone of user input. Chatbots can use this information to tailor responses based on the user's mood or sentiment.

PYTHON PROGRAM:

```
    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

df.drop(columns=['answer tokens','question tokens'],axis=1,inplace=True)
df['encoder_inputs']=df['question'].apply(clean_text)
df['decoder_targets']=df['answer'].apply(clean_text)+' <end>'
df['decoder_inputs']='<start> '+df['answer'].apply(clean_text)+' <end>'

# Load the spaCy language model

nlp = spacy.load("en_core_web_sm")


# Define a function to process user input

def process_user_input(user_input):
```

```
    # Tokenize and parse the user input using spaCy
    doc = nlp(user_input)

    # Extract named entities (e.g., names, places, organizations)
    entities = [(ent.text, ent.label_) for ent in doc.ents]

    # Extract nouns and verbs
    nouns = [token.text for token in doc if token.pos_ == "NOUN"]
    verbs = [token.text for token in doc if token.pos_ == "VERB"]

    return {
        "tokens": [token.text for token in doc],
        "entities": entities,
        "nouns": nouns,
        "verbs": verbs,
    }

# Example usage:
user_input = "Tell me about OpenAI, and recommend a book."
result = process_user_input(user_input)

# Display the processed information
print("User Input Tokens:", result["tokens"])
print("Named Entities:", result["entities"])
print("Nouns:", result["nouns"])
print("Verbs:", result["verbs"])
```

OUTPUT:

	answer	encoder_inputs	decoder_targets	decoder_inputs	
0	hi, how are you doing?	i'm fine. how about yourself?	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.	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?	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?	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.	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?	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.	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?	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.	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.	it ' s okay . it ' s a really big campus .	good luck with school . <end>	<start> good luck with school . <end>

RESPONSES:

In the context of a chatbot, a "response" refers to the reply or answer that the chatbot provides to a user's input or query. Responses are a fundamental aspect of the chatbot's interaction with users, and they play a crucial role in delivering a satisfying user experience. Here are key points to understand about responses in chatbots:

- **Purpose of Responses:** Responses in a chatbot serve several purposes, including providing information, answering questions, offering assistance, guiding users, and engaging in a conversation. The purpose of a response depends on the user's input and the chatbot's functionality.
- **User Input:** Responses are generated in reaction to the user's input, which can be in the form of text, voice, or other forms of communication, depending on the chatbot's design.
- **Response Types:**
  - **Accurate Answers:** Chatbots should aim to provide accurate and relevant answers to user queries. For factual questions, the response should contain precise information.
  - **Suggestions:** Chatbots can offer suggestions or recommendations based on user preferences or historical interactions. For example, suggesting products, movies, or articles.
  - **Assistance:** Responses can include instructions, guidance, or step-by-step assistance to help users complete tasks or solve problems.
  - **Engagement:** In more conversational chatbots, responses may focus on engaging the user, sharing jokes, or maintaining a friendly and interactive tone.
  - **Error Handling:** When users provide unclear or incorrect input, the chatbot's response should be designed to handle errors gracefully, asking for clarification or providing helpful suggestions

PYTHON PROGRAM:

```
# Define a function to generate responses
```

```
def generate_response(user_input):
    user_input = user_input.lower()  # Convert user input to lowercase for case-insensitivity

    # Define a dictionary of responses based on user input
    responses = {
        "hello": "Hello! How can I assist you today?",
        "how are you": "I'm just a computer program, so I don't have feelings, but thanks for asking!",
        "what is your name": "I am a chatbot.",
        "tell me a joke": "Why don't scientists trust atoms? Because they make up everything!",
        "recommend a movie": "Sure! What genre are you interested in?",
        "recommend a book": "Of course! What genre or author do you prefer?",
    }

    # Check if the user input matches any predefined responses
    if user_input in responses:
        return responses[user_input]
    else:
        # If there's no predefined response, provide a generic response
        return "I'm not sure how to respond to that. Please ask me a different question."

# Example usage:
user_input = input("You: ")  # Get user input
bot_response = generate_response(user_input)
print("Chatbot:", bot_response)
```

INTEGERATION:

Integration in the context of a chatbot refers to the process of connecting the chatbot with other systems, platforms, or services to enhance its functionality and usability. Chatbot integration allows it to interact with external data sources, perform tasks, and provide valuable services to users. Here are key aspects to understand about integration in chatbots:

- **Platform Integration:** Chatbots can be integrated into various platforms or channels,

- **API Integration:** Chatbots can connect to external APIs (Application Programming Interfaces) to access data or services from third-party sources. For example, a chatbot may integrate with weather APIs to provide weather forecasts or with e-commerce APIs to retrieve product information and prices.
- **Database Integration:** Chatbots can access and query databases to retrieve and update information. For instance, a chatbot for an e-commerce website might connect to a product database to fetch product details.
- **Machine Learning and AI Integration:** Advanced chatbots can integrate with machine learning and AI models to perform tasks like sentiment analysis, language translation, or content recommendation. These models may be hosted externally or developed in-house.
- **Authentication and Security:** Integration often involves handling authentication and security measures to ensure that the chatbot can access external systems securely and protect user data.
- **Data Synchronization:** In scenarios where chatbots are used in tandem with other applications, integration may involve data synchronization to ensure that data remains consistent across systems.

- **User Authentication:** For personalized experiences, chatbots may integrate with user authentication systems to identify and authenticate users, enabling tailored interactions and access to user-specific data.

- **Transaction Processing:** Some chatbots can integrate with payment gateways and financial systems to facilitate transactions, such as making bookings, purchases, or payments.
- **Web Scraping:** In cases where official APIs are unavailable, chatbots may employ web scraping techniques to extract information from websites.
- **Continuous Integration and Deployment (CI/CD):** Integration pipelines can be set up to streamline the development, testing, and deployment of chatbot updates and improvements.
- **Analytics and Monitoring:** Integration with analytics tools allows for the collection of data on user interactions, enabling the evaluation of chatbot performance and user behaviour.
- **Feedback Integration:** Chatbots can collect and integrate user feedback into their design and improvement processes, ensuring a more user-centric experience.

Overall, integration is a critical aspect of chatbot development, as it expands the chatbot's capabilities and allows it to provide valuable services and information from various sources. Effective integration strategies should be carefully planned and implemented to ensure that the chatbot functions seamlessly within its intended ecosystem.

TESTING AND IMPROVEMENT:

Testing and improvement in chatbots are crucial processes:

Testing:

- Ensures the chatbot functions correctly.
- Involves functional, user, regression, performance, security, and NLP testing.
- Identifies issues and verifies usability.

Improvement:

- Refines chatbot based on user feedback.
- Iterates on design, responses, and functionality.
- Incorporates analytics, A/B testing, and error handling.
- Keeps content updated and enhances personalization, naturalness, and security.
- Ensures continuous evolution and user satisfaction.

PYTHON PROGRAM:

```
# Sample dictionary to store user feedback
user_feedback = {
    "positive": [],
    "negative": [],
    "suggestions": [],
}


# Function to collect and process user feedback
def collect_user_feedback(feedback):
    feedback_type = feedback.get("type", None)
    if feedback_type is None:
        return "Please provide feedback type ('positive', 'negative', or 'suggestion')."

    message = feedback.get("message", "")
    if feedback_type == "positive":
        user_feedback["positive"].append(message)
    elif feedback_type == "negative":
        user_feedback["negative"].append(message)
    elif feedback_type == "suggestion":
        user_feedback["suggestions"].append(message)
```

```
    return "Thank you for your feedback!"

# Function to analyze user feedback and make improvements
def analyze_and_improve():
    # Analyze user feedback and chat logs here
    # Implement logic to identify areas for improvement
    # Adjust chatbot responses or functionality accordingly

    # For simplicity, let's assume some improvements are made
    improved_responses = {
        "positive": ["Thank you!", "Great to hear that!"],
        "negative": ["I apologize for the inconvenience.", "We'll work on improving."],
    }

    # Update chatbot responses based on improvements
    # In practice, this would involve modifying your chatbot's response generation logic
    chatbot_responses.update(improved_responses)

# Example user feedback
user_feedback_data = [
    {"type": "positive", "message": "I found the chatbot very helpful."},
    {"type": "negative", "message": "The chatbot misunderstood my question."},
    {"type": "suggestion", "message": "Can you provide more examples?"}
]

# Collect and process user feedback
for feedback in user_feedback_data:
    response = collect_user_feedback(feedback)
    print(response)
```

```
# Analyze user feedback and make improvements
analyze_and_improve()

# Example chatbot responses after improvements
chatbot_responses = {
    "positive": ["Hello! How can I assist you today?", "Sure, I can help with that."],
    "negative": ["I apologize for the inconvenience.", "Let me try to better understand your question."],
}

# Simulate chatbot interaction
user_input = "Can you help me with a math problem?"
print("User: " + user_input)
print("Chatbot: " + random.choice(chatbot_responses["positive"]))

# You would continue collecting user feedback and iterating on improvements as needed.
```

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.