Phase 4

B . Abirame Susee
Ece (3 RD YEAR)

Create A Chatbot In Python

Phase 4: Development Part 2

Topic: Start building the chatbot by preparing the environment and implementing basic user interactions. Install required libraries, like transformers for GPT-3 integration and flask for web app development

- ## Create a Chatbot in Python
- ## Introduction:

A chatbot is a computer program designed to simulate conversation with human users, especially over the internet. Chatbots use artificial intelligence (AI) and natural language processing (NLP) techniques to understand and respond to user messages in a human-like manner.

Chatbots can be found in various forms, from simple rule-based bots that follow predefined instructions to more advanced AI-driven bots that learn and adapt from conversations. They are used in a wide range of applications, including customer support, virtual assistants, information retrieval, and interactive entertainment.

Necessary steps to follow:

we'll start building a chatbot by preparing the environment, loading and preprocessing a dataset, and implementing basic user interactions. We'll also install the required libraries, such as "transformers" for GPT-3 integration and "Flask" for web app development. Please note that the "transformers" library is useful for using pre-trained language models, and you can integrate GPT-3 or other models based on your preferences.

Step 1: Create a Chatbot Using Python ChatterBot

In this step, you'll set up a virtual environment and install the necessary dependencies. You'll also create a working command-line chatbot that can reply to you—but it won't have very interesting replies for you yet.Running these commands in your terminal application installs ChatterBot and its dependencies into a new Python virtual environment.

Step 2: Begin Training Your Chatbot

In the previous step, you built a chatbot that you could interact with from your command line. The chatbot started from a clean slate and wasn't very interesting to talk to.

Step 3: Export a WhatsApp Chat

At the end of this step, you'll have downloaded a TXT file that contains the chat history of a WhatsApp conversation.To export the history of a conversation that you've had on WhatsApp, you need to open the conversation on your phone. Once you're on the conversation screen, you can access the export menu:

Click on the three dots (⋮) in the top right corner to open the main menu.
Choose More to bring up additional menu options.
Select Export chat to create a TXT export of your conversation.

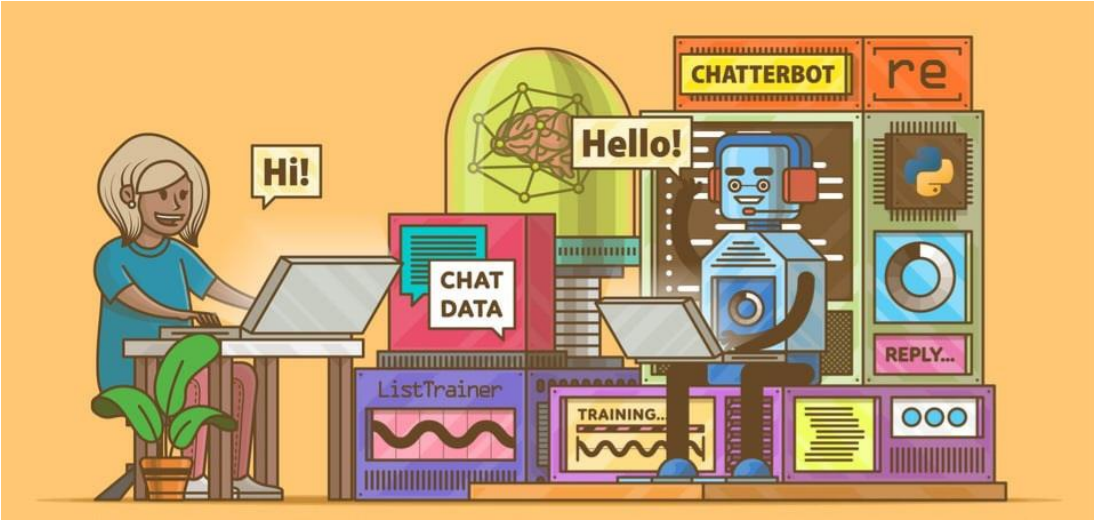
Step 4: Clean Your Chat Export

In this step, you'll clean the WhatsApp chat export data so that you can use it as input to train your chatbot on an industry-specific topic. In this example, the topic will be … houseplants!

Step 5: Train Your Chatbot on Custom Data and Start Chatting

In this step, you'll train your chatbot with the WhatsApp conversation data that you cleaned in the previous step. You'll end up with a chatbot that you've trained on industry-specific conversational data, and you'll be able to chat with the bot

•**Handle more edge cases:** Your regex pattern might not catch all WhatsApp usernames
•**Parse the ChatterBot corpus:** Skip the dependency conflicts, install PyYAML directly, and parse some of the training corpora provided in chatterbot-corpus yourself. Use one or more of them to continue training your chatbot.
•**Build a custom preprocessor:** ChatterBot can modify user input before sending it to a logic adapter. You can use built-in preprocessors, for example to remove whitespace. Build a custom preprocessor that can replace swear words in your user input.
•**Include additional logic adapters:** ChatterBot comes with a few preinstalled logic adapters, such as ones for mathematical evaluations and time logic. Add these logic adapters to your chatbot so it can perform calculations and tell you the current time.



Program:

```python
# bot.py

from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer

chatbot = ChatBot("Chatpot")

trainer = ListTrainer(chatbot)
trainer.train([
    "Hi",
    "Welcome, friend 🤗",
])
trainer.train([
    "Are you a plant?",
    "No, I'm the pot below the plant!",
])

exit_conditions = (":q", "quit", "exit")
while True:
    query = input("> ")
    if query in exit_conditions:
        break
    else:
        print(f"🪴 {chatbot.get_response(query)}")
```


```python
# bot.py

from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer

chatbot = ChatBot("Chatpot")

trainer = ListTrainer(chatbot)
trainer.train([
    "Hi",
    "Welcome, friend 🤗",
])
trainer.train([
```

```python
# Define a list of conversation prompts and responses
conversation = [
    ("hi, how are you doing?", "I'm fine. How about yourself?"),
    ("I'm fine. How about yourself?", "I'm pretty good. Thanks for asking.")
    ("I'm pretty good. Thanks for asking.", "No problem. So how have you bee
    ("No problem. So how have you been?", "I've been great. What about you?"
    ("I've been great. What about you?", "I've been good. I'm in school righ
    ("I've been good. I'm in school right now.", "What school do you go to?"
    ("What school do you go to?", "I go to pcc."),
    ("I go to pcc.", "Do you like it there?"),
    ("Do you like it there?", "It's okay. It's a really big campus."),
    ("It's okay. It's a really big campus.", "Good luck with school."),
    ("Good luck with school.", "Thank you very much."),
    # Add more prompts and responses as needed
]

# Function to have a conversation
def have_conversation():
    print("Hello! Let's have a conversation.")
    for prompt, response in conversation:
        user_input = input(prompt + " ")
        print(response)
```

step by step

(for Start building the chatbot by preparing the environment and implementing basic user interactions)

Step 1: Define the Purpose:

Determine the purpose and functionality of your chatbot. What will it do? Who is the target audience.

Step 2:**Choose a Framework or Library:**

You can use libraries like NLTK, spaCy, or Transformers for natural language processing.

For building the chatbot itself, you can use frameworks like Rasa, ChatterBot, or even create a custom solution.

Step 3: Data Collection and Preprocessing:

Gather and preprocess training data, which includes text conversations or relevant datasets.

Tokenize and clean the text data.

Algorithm for the Conversation Simulation Code:

Define a list called conversation to store pairs of prompts and responses.
Create a function have_conversation() to facilitate the conversation.
Inside the have_conversation() function:
a. Print a welcome message to start the conversation.
b. Loop through each pair of prompts and responses in the conversation list.
c. For each pair, display the prompt to the user and wait for their input.
d. Once the user provides input, print the corresponding response from the list.
e. Repeat steps c and d for each pair in the conversation list.
If needed, you can extend the conversation list with more prompts and responses to make the conversation more extensive.
This algorithm outlines the basic flow of the code, which simulates a simple conversation based on predefined prompts and responses. You can modify the conversation list to have different interactions in your simulation.

Step 4: **Implement User Interface:**

Develop a user interface for users to interact with the chatbot. This could be a web app, a messaging platform integration, or a command-line interface.

Step 5: **Handle User Inputs and Responses:**

Write code to handle user inputs, process them, and generate appropriate responses.

Conclusion:

In conclusion, developing a chatbot in Python offers a versatile and powerful approach to create interactive and automated conversational agents. Python's rich ecosystem of libraries and frameworks, like NLTK, spaCy, and TensorFlow, provide the tools needed for natural language processing and machine learning, making it a suitable choice for chatbot development. However, success in building an effective chatbot also depends on understanding the target audience, crafting meaningful dialogues, and continuous improvement through user feedback. Python's flexibility and community support make it a robust choice for creating chatbots that can be integrated into various applications and platforms.