CREATE CHATBOT IN PYTHON

PHASE 5 : PROJECT SUBMISSSION

TOPIC : COMPLETE PROJECT PREPARATION FOR SUBMISSION

puzzel.

INTRODUCTION

Chatbots are revolutionizing user interaction. Python provides a powerful platform for building effective chatbots. In this presentation, we will explore the basics of building chatbots in Python, and learn how to create chatbots that can interact with users in a natural and intuitive way.

- **Choose a Chatbot Framework:** There are several Python frameworks and libraries available for building chatbots. Some popular options include ChatterBot, NLTK, spaCy, and Rasa. Select the one that best suits your project's requirements and your familiarity with the framework.
- **Install Dependencies:** Once you've chosen a framework, you'll need to install any required dependencies and libraries. You can typically use pip, the Python package manager, to install these packages.
- **Data Collection and Preprocessing:** To create a chatbot, you need a dataset of conversations, or you can create your own.

This data will be used for training your chatbot. Preprocess the data by cleaning and formatting it for training purposes.

- **Model Training:** Utilize your chosen framework to train your chatbot model on the prepared dataset. This step involves natural language processing (NLP) techniques, such as tokenization, stemming, and entity recognition.
- **Integration:** After training, you can integrate your chatbot into a platform or application. This may involve using web APIs or frameworks like Flask or Django for web-based chatbots. For integration with messaging platforms like Facebook Messenger or Slack, you'll need to set up the appropriate API connections.
- **User Interface:** Design and implement the user interface for your chatbot. Depending on your application, this might be a web-based interface, a mobile app, or integration with an existing messaging platform.
- **Testing and Refinement:** Thoroughly test your chatbot to ensure it can understand and respond effectively to user

queries. This step may require fine-tuning the model, improving the conversation flow, and addressing any issues or limitations.

- **Deployment:** Once your chatbot is ready, you can deploy it to a server or cloud platform to make it accessible to users.
- Monitoring and Maintenance: Continuously monitor the chatbot's performance and gather user feedback to improve its responses and capabilities. Regular updates and maintenance are crucial for keeping your chatbot relevant and useful.
- **Scaling:** If your chatbot becomes popular and experiences increased usage, be prepared to scale your infrastructure to handle the load.

PROJECT DEFINION

Chatbots are computer programs designed to simulate conversation with human users. They can be used to automate tasks, provide customer support, and even entertain users. Chatbots can be built using a variety of programming languages, but Python is a popular choice due to its simplicity and versatility.

Chatbots can serve various purposes, including answering frequently asked questions, assisting with tasks, offering customer support, providing information, and even engaging in casual conversation. They are employed in a wide range of applications, such as customer service, e-commerce, virtual assistants, healthcare, and more.

Chatbots use natural language processing (NLP) techniques and machine learning algorithms to understand and generate human language. They can be rule-based, where they follow a predefined set of rules to respond, or they can be AI-driven, using deep learning and neural networks to learn from data and improve their responses over time.

WHY PYTHON?

Python is a popular programming language for building chatbots due to its simplicity, ease of use, and versatility. Python has a large number of libraries and frameworks that make it easy to build chatbots quickly and efficiently. Additionally, Python has a large and active community of developers who contribute to the development of chatbot tools and resources.

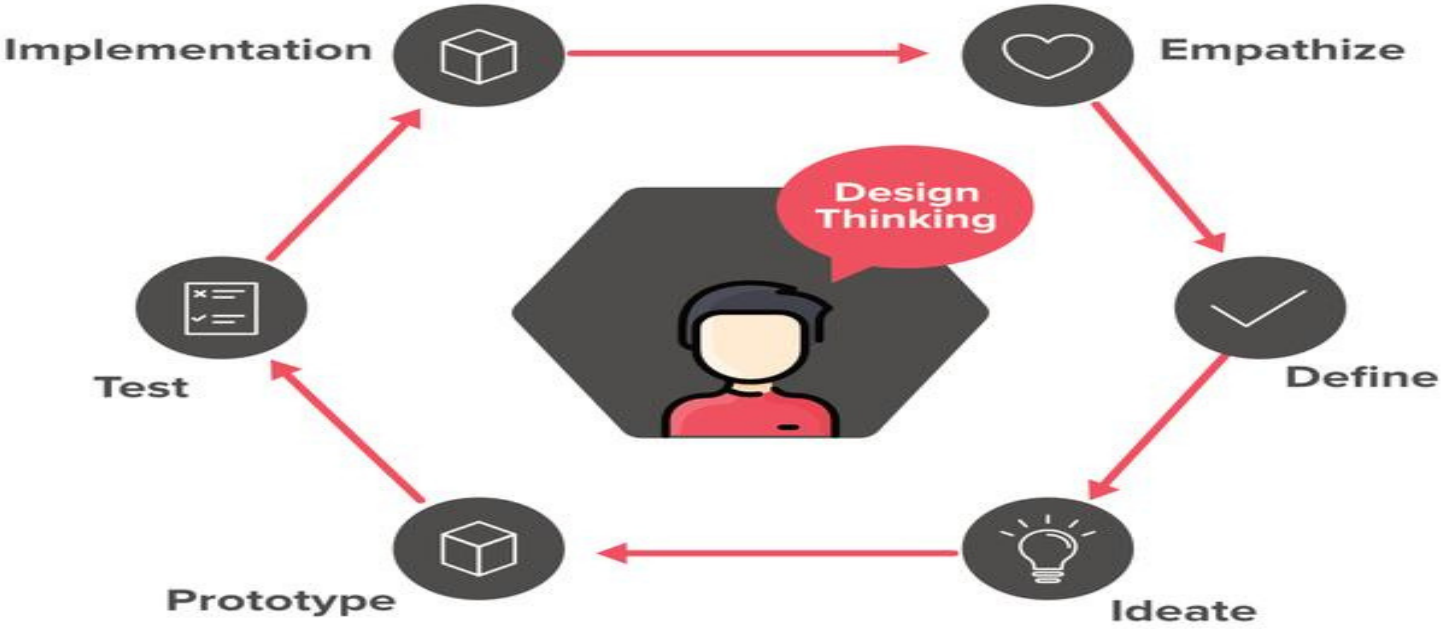
- **Ease of Learning and Readability:** Python is known for its simple and readable syntax. This makes it an excellent language for beginners and experienced developers alike. Building and maintaining chatbots in Python is more accessible, which accelerates development and reduces the chances of errors.
- **Rich Ecosystem of Libraries:** Python has a vast ecosystem of libraries and frameworks that are well-suited for natural language processing (NLP) and machine learning, which are crucial components of chatbot development. Libraries like NLTK, spaCy, and scikit-learn provide powerful tools for text analysis and machine learning tasks.
- **Machine Learning and NLP Tools:** Python has extensive support for machine learning and NLP, with libraries such as TensorFlow, PyTorch, and Gensim. These tools enable developers to implement sophisticated algorithms for language understanding, sentiment analysis, and text generation, which are essential for creating intelligent chatbots.

- **Community Support:** Python boasts a large and active community of developers. You can find a wealth of online resources, forums, and tutorials that can help you troubleshoot issues, find solutions, and stay updated with best practices in chatbot development.
- **Open Source Frameworks:** Python offers open-source chatbot frameworks and libraries like ChatterBot, Rasa, and Dialogflow, which expedite the development process. These frameworks come with pre-built functionalities for natural language understanding, conversation management, and integration with various platforms.
- **Platform-Agnostic:** Python is platform-agnostic, meaning that chatbots created in Python can run on various operating systems without modification. This versatility is useful if you plan to deploy your chatbot across different environments.
- **Cross-Domain Applicability:** Python is versatile and can be applied to a wide range of use cases. Whether you're creating a customer support chatbot, a virtual assistant, a

recommendation system, or any other application, Python's flexibility allows you to adapt it to your specific requirements.

- **Scalability:** Python can be used to develop chatbots that are easy to scale. If your chatbot's user base grows, you can deploy it on cloud platforms or distributed systems without significant reengineering.
- **Robust Ecosystem of Deployment Options:** Python chatbots can be deployed on various platforms, including web applications, mobile apps, messaging platforms (e.g., Facebook Messenger, Slack), and more. Python's versatility ensures that you can reach your target audience through the appropriate channels.

DESIGN THINKING PROCESS



Natural Language Processing (NLP) plays a crucial role in creating chatbots in Python, as it enables them to understand and generate human language. Here are the key components and steps involved in using NLP for chatbot development in Python:

- **Text Preprocessing:**

•**Tokenization:** Break text into individual words or tokens. Python libraries like NLTK, spaCy, and scikit-learn offer tokenization tools.

• **Stopword Removal:** Filter out common words (stopwords) like "and," "the," "in" that don't carry significant meaning for chatbot understanding.

• **Stemming and Lemmatization:** Reduce words to their base form (stemming) or dictionary form (lemmatization) to simplify text analysis. NLTK and spaCy provide stemmers and lemmatizers.

• **Named Entity Recognition (NER):**

• Identify and categorize entities in text, such as names of people, organizations, locations, dates, and more. Libraries like spaCy and NLTK offer NER capabilities.

• **Text Classification:**

• Train machine learning models to classify text into predefined categories or intents. This helps the chatbot understand user input and respond accordingly. Popular classification algorithms are available in libraries like scikit-learn and spaCy.

• **Intent Recognition:**

- Determine the user's intent based on their input. Intent recognition is essential for routing user queries to the appropriate responses. Frameworks like Rasa provide dedicated tools for intent recognition.
- **Sentiment Analysis:**
  - Assess the sentiment of user messages to gauge their emotions or opinions. Sentiment analysis libraries like TextBlob and VADER are helpful in this context.
- **Dialog Management:**
  - Create a conversation flow and context management system to maintain the context of ongoing interactions. Frameworks like Rasa and Dialogflow offer capabilities for managing dialog flow.
- **Response Generation:**
  - Based on the recognized intent and context, generate appropriate responses. Response generation can be rule-based or use more advanced techniques like sequence-to-sequence models for natural-sounding replies.
- **Machine Learning and Deep Learning:**

- Implement machine learning or deep learning models for more advanced NLP tasks. Popular libraries like TensorFlow and PyTorch provide tools for training custom models for chatbot development.

- **Chatbot Frameworks:**
  - Consider using chatbot development frameworks like Rasa, ChatterBot, or Dialogflow. These frameworks often include pre-built components for NLP and dialog management, streamlining the development process.
- **User Testing and Feedback:**
- Continuously test your chatbot with real users and gather feedback to refine its NLP capabilities and improve its responses.
- **Integration:**
  - Integrate your NLP-powered chatbot into the desired platforms or channels, such as websites, mobile apps, messaging apps, or other communication channels.
- **Scalability and Deployment:**
  - Prepare your chatbot for deployment on the chosen infrastructure, whether it's a local server or cloud platform. Ensure that it can handle increased usage as needed.

When creating a chatbot in Python, you have a wealth of NLP libraries and tools at your disposal, making it easier to implement the necessary natural language processing components. The specific NLP techniques and tools you choose will depend on the complexity of your chatbot's requirements and the available resources for development.

ALGORITHM

Step 1:Start the program
Step 2:Using vscode,we compiled the code and run the code
Step 3:In Github,we upload the file
Step 4:We created README file,which describes about our project
Step 5:We copied the link and passed in our project Step 6:We successfully submitted our projecT
Step 7:Stop the program

PROGRAM

#bot.py

```
2
3from chatterbot import ChatBot
4
5chatbot = ChatBot("Chatpot")
6
7exit_conditions = (":q", "quit", "exit")
8while True:
9 query = input("> ")
10 if query in exit_conditions:
11 break
12 else:
13 print(f" {chatbot.get_response(query)}")
```

OUTPUT

```
 > hello
hello
> are you a plant?
hello
```

> can you chat, pot?
hello

ABSTRACT

Chatbots, or conversational interfaces as they are also known, present a new way for individuals to interact with computer systems. Traditionally, to get a question answered by a software program involved using a search engine, or filling out a form. A chatbot allows a user to simply ask questions in the same manner that they would address a human. The most well known chatbots currently are voice chatbots: Alexa and Siri. However, chatbots are currently being adopted at a high rate on computer chat platforms. The technology at the core of the rise of the chatbot is natural language processing ("NLP"). Recent advances in machine learning have greatly improved the accuracy and effectiveness of natural language processing, making chatbots a viable option for many organizations. This improvement in NLP is firing a great deal of additional research which should lead to continued improvement in the effectiveness of chatbots in the years to come.

TRAINING THE CHATBOT

In the previous step, you built a chatbot that you could interact with from your command line. The chatbot started from aclean slate and wasn't very interesting to talk to. In this step, you'll train your chatbot using ListTrainer to make it a little smarter from the start. You'll also learn about built-in trainers that come with ChatterBot, including their limitations. Your chatbot doesn't have to start from scratch, and ChatterBot provides you with a quick way to train your bot. You'll use ChatterBot's ListTrainer to provide some conversation samples that'll give your chatbot more room to grow:

```
# 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)}")
```

TRAINED CHATBOT OUTPUT

> hi

Welcome, friend 🤗

> hello

are you a plant?

> me?

are you a plant?

> yes

hi >
are you a plant?
No, I' m the pot below the plant!
> cool
Welcome, friend 🤗

EVALUATING THE PERFORMANCE FOR CHATBOT

1. Activation rate
2. Average Session duration
3.Session per user
4. Voluntary User Engagement
5. Retention Rate
6. Goal Completion Rate (GCR)
7. Revenue growth
8. Confusion Rate
9. Human Fallback rate
10. Conversion sentiment

11. Artificial Intelligence and Machine Learning rate

INTRODUCTION STATEMENT FOR CHATBOT

1. Introduction Briefly introduce the concept of a chatbot and its significance in various industries. Highlight the objective of this project, which is to develop a functional chatbot using Python.
2. Target Audience Define the intended users or audience for the chatbot. Specify the domain or industry where the chatbot will be deployed (e.g., customer support, e-commerce, healthcare).
3.Purpose and Scope State the primary purpose of the chatbot (e.g., answering FAQs, providing recommendations, processing orders). Define the boundaries of the chatbot's capabilities and limitations (e.g., no personal information handling, no transaction processing).
4. Functional Requirements Specify the core functionalities the chatbot is expected to perform: Natural Language Understanding (NLU) Response Generation Integration with external systems (if applicable) Error handling and fallback mechanisms
5. Technical Stack Specify the technologies and libraries that will be used in the development of the chatbot: Programming Language: Python NLP Library: NLTK, spaCy, or similar Additional

libraries for web frameworks, if applicable PROBLEM STATEMENT FOR CHATBOT

6. Architecture Provide a high-level overview of the chatbot's architecture: Input processing Natural Language Understanding (NLU) Logic processing Response generation Integration points (if any)

7. User Experience (UX) Design Describe the expected user interactions and the flow of conversations. Optionally, include mockups or wireframes to illustrate the user interface.

8. Data Requirements Specify any datasets or corpora needed for training NLP models (if applicable). Discuss data privacy and security considerations, especially if handling sensitive information .

9. Testing and Evaluation Define the testing criteria and methodologies to assess the chatbot's performance. Include sample test cases and expected outcomes.

10. Deployment and Scaling Discuss deployment options (e.g., web-based, messaging platforms, mobile apps). Consider scalability aspects for potential future growth.

11. Timeline and Milestones Provide a rough timeline with key milestones for development, testing, and deployment.

12. Budget and Resource Allocation Outline any budget considerations, if applicable (e.g., cloud hosting costs, licensing fees for external services). Allocate resources (human and technical) required for the project.

13. Risks and Mitigations Identify potential risks or challenges that could arise during the development process. Propose strategies to mitigate these risks.

14. Ethical Considerations Address ethical concerns, such as bias in NLP models, data privacy, and transparency in AI-powered interactions.

15. Conclusion Summarize the key points of the problem statement and reiterate the importance of the project. This problem statement outline provides a structured framework for developing a chatbot in Python, encompassing various aspects from technical requirements to ethical considerations. It serves as a guide to ensure clarity and alignment in the development process.

MODULE

Nobody likes to be alone always, but sometimes loneliness could be a better medicine to hunch the thirst for a peaceful environment. Even during such lonely quarantines, we may ignore humans but not humanoids. Yes, if you have guessed this article for a chatbot, then you have cracked it right. We won't require 6000 lines of code to create a chatbot but just a six-letter word "Python" is enough. Let us have a quick glance at Python's ChatterBot to create our bot. ChatterBot is a Python library built based on machine learning with an inbuilt conversational dialog flow and training engine. The bot created using this library will get trained automatically with the response it gets from the user.

FEATURE EXTRACTION TECHNIQUES

1.Bag-of-Words (BoW): BoW represents text data by creating a vocabulary of all unique words in the dataset. Each document is then represented as a vector, where each element corresponds to the frequency or presence of a word in the vocabulary. Libraries like CountVectorizer in scikit-learn can be used for this technique.

```
from sklearn.feature_extraction.text import CountVectorizer
# Create an instance of CountVectorizer
vectorizer = CountVectorizer()
# Fit and transform the data
X = vectorizer.fit_transform(text_data)
```

2.Term Frequency-Inverse Document Frequency (TF-IDF): TF-IDF represents the importance of a word in a document relative to its frequency across all documents. It is a product of Term Frequency (TF) and Inverse Document Frequency (IDF). TF-IDF can be implemented using the TfidfVectorizer in scikit-learn.

```
from sklearn.feature_extraction.text import TfidfVectorizer
# Create an instance of TfidfVectorizer
vectorizer = TfidfVectorizer()
# Fit and transform the data
X = vectorizer.fit_transform(text_data)
```

3.Word Embeddings (e.g., Word2Vec, GloVe):Word embeddings represent words as dense vectors in a continuous vector space.These vectors capture semantic relationships between

words.Libraries like Gensim provide tools for working with Word2Vec.

```
from gensim.models import Word2Vec
# Train a Word2Vec model
            model = Word2Vec(sentences, vector_size=100, window=5,
min_count=1, sg=0)
# Get the embedding for a word
embedding = model.wv['word']
```

INNOVATIVE TECHNIQUES FOR CHATBOT IN PYTHON

Generative Pre-trained Transformer 3 (GPT-3): Utilize models like GPT-3, which are capable of generating human-like text and can be fine-tuned for specific tasks. OpenAI' s GPT-3 can be accessed through the OpenAI API.

```
# Example of GPT-3 API usage with OpenAI import openai
response = openai.Completion.create( engine= "davinci-codex ",
Prompt= "Translate the following English text to French: '{}'",
max_tokens=60 )
```

2.Transformer-based Chatbot Models: Build chatbots using advanced transformer models like BERT, GPT-2, or T5, which have shown remarkable performance in natural language understanding and generation tasks.

```
from transformers import pipeline
# Example of using a transformer for text generation
generator = pipeline('text-generation ',
model= 'EleutherAI/gpt-neo-2.7B')
generated_text = generator("Once upon a time ",
max_length=50, do_sample=True)[
```

3.Multi-modal Chatbots: Combine text-based interactions with other modalities like images, videos, or voice for a richer user experience. Utilize libraries like OpenCV for image processing and speech recognition libraries for voice interactions.

```
# Example of using OpenCV for image processing import cv2
# Example of using a speech recognition library import
speech_recognition as sr
```

DATA PREPROCESSING STEPS

Data preprocessing is a crucial step when building a chatbot in Python, as it helps ensure that the input data is clean, structured, and ready for natural language processing. Below are the common data preprocessing steps for chatbot development: 1.Data Collection: Gather relevant data from various sources, such as text corpora, databases, or APIs. Ensure that the data aligns with the chatbot's intended purpose and scope.

2.Text Cleaning: Remove HTML tags, special characters, and non-textual elements from the data. Convert the text to lowercase to ensure consistency.

3.Tokenization: Tokenize the text into words or subword units (e.g., using libraries like NLTK or spaCy). This step breaks the text into meaningful units, making it easier for the chatbot to process.

4. Stopword Removal: Remove common stopwords (e.g., "and," "the," "is") that do not contribute significantly to the meaning of the text. This reduces the dimensionality of the data.

5. Lemmatization or Stemming: Apply lemmatization or stemming to reduce words to their base form. This helps in treating

different forms of a word as the same (e.g., "running" becomes "run").

6. Spell Checking and Correction: Implement a spell checker to correct common typos and improve user experience.

7. Handling Abbreviations and Acronyms: Expand common abbreviations and acronyms to improve user understanding. Data Formatting: Format the data in a structured way (e.g., JSON or CSV) for easy retrieval and processing.

8. Handling Synonyms: Create a dictionary of synonyms to ensure that different words with similar meanings are treated consistently.

9. Data Labeling (if supervised learning): Annotate or label data with relevant tags or intents, especially for training a supervised learning-based chatbot.

10. Data Splitting (if supervised learning):Divide the data into training, validation, and test sets to train and evaluate the chatbot's performance.

11. Data Vectorization:Convert the text data into numerical vectors, such as TF-IDF (Term Frequency-Inverse Document

Frequency) or word embeddings (Word2Vec, GloVe).These vectors are used as inputs for machine learning or deep learning models.

12.Handling Imbalanced Data:If there's an imbalance in the distribution of classes or intents, consider techniques like oversampling, undersampling, or generating synthetic data to balance the datas.

DESIGN IDEAS FOR CHATBOT IN PYTHON

Even if you keep running your chatbot on the CLI for now, there are many ways that you can improve the project and continue to learn about the ChatterBot library:

1. Handle more edge cases: Your regex pattern might not catch all WhatsApp usernames. You can throw some edge cases at it and improve the stability of your parsing while building tests for your code.

2. Improve conversations: Group your input data as conversations so that your training input considers consecutive messages sent by the same user within an hour a single message. 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.

3. 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.

4. 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.

5. Write a custom logic adapter: Create a custom logic adapter that triggers on specific user inputs, for example when your users ask for a joke.

6. Incorporate an API call: Build a logic adapter that can interact with an API service, for example by repurposing your weather CLI project so that it works within your chatbot.

DATASET FOR CHATBOT IN PYTHON

LINK FOR DATASET:

https://www.kaggle.com/dataset/grafstor/simple-dialogs-for-chatbot

hi, how are you doing? i'm fine. how about yourself? i'm fine. how about yourself? i'm prettygood. thanks for asking. i'm pretty good. thanks for asking. no problem. so how have you been? 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 right now. 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. how'sit going? i'm doing well. how about you? i'm doing well. how about you? never better, thanks. never better, thanks. so how have you been lately? so how have you been lately? i've actually been pretty good. you? i've actually been pretty good. you? i'm actually in school right now. i'm actually in school right now. which school do you attend? which schooldo you attend? i'm attending pcc right now. i'm attending pcc right now. are you enjoying it there? are you enjoying it there? it's not

bad. there are a lot of people there. it's not bad. there are a lot of people there. good luck with that. good luck with that. thanks. how are you doing today? i'm doing great. what about you? i'm doing great. what about you? i'm absolutely lovely, thank you. i'm absolutely lovely, thank you. everything's been good with you? everything's been good with you? i haven't been better. how about yourself? i haven't been better. how about yourself? i started school recently. i started school recently. where are you going to school? where are you going to school? i'm going to pcc. i'm going to pcc. how do you like it so far? how do you like it so far? i like it so far. my classes are pretty good right now. i like it so far. my classes are pretty good right now. i wish you luck. it's an ugly day today. i know. i think it may rain.

i know. i think it may rain. it's the middle of summer, it shouldn't rain today. it's the middle of summer, it shouldn't rain today. that would be weird. that would be weird. yeah, especially since it's ninety degrees outside. yeah, especially since it's ninety degrees outside. i know, it would be horrible if it rained and it was hot outside. i know, it would be horrible if it rained and it was hot

outside. yes, it would be. yes, it would be. i really wish it wasn't so hot every day. i really wish it wasn't so hot everyday. me too. i can't wait until winter. me too. i can't wait until winter. i like winter too, but sometimes it gets too cold. i like winter too, but sometimes it gets too cold. i'd rather be cold than hot. i'd ratherbe cold than hot. me too. it doesn't look very nice outside today. you're right. i think it's going to rainlater. you're right. i think it's going to rain later. in the middle of the summer, it shouldn't be raining. in the middle of the summer, it shouldn't be raining. that wouldn't seem right. that wouldn't seem right. considering that it's over ninety degrees outside, that would be weird. considering that it's over ninety degrees outside, that would be weird. exactly, it wouldn't be nice if it started raining. it's too hot. exactly, it wouldn't be nice if it started raining. it's too hot. i know, you're absolutely right. i know, you're absolutely right. i wish it would cool off one day. i wish it would cool off one day. that's how i feel, i want winter to come soon. that's how i feel, i want winter to come soon. i enjoy the winter, but it gets really cold sometimes. i enjoy the winter, but it gets really cold sometimes. i know what you mean, but i'd rather be cold than hot.

i know what you mean, but i'd rather be cold than hot. that's exactly how i feel. i wish it was a nicer day today. that is true. i hope it doesn't rain.

CONCLUSION

Chatbots are revolutionizing user interaction, and Python provides a powerful platform for building effective chatbots. By leveraging the power of NLP and conversation design, developers can create chatbots that provide a natural and intuitive user experience. With the right skills and knowledge, anyone can build a chatbot in Python