# PHASE 4: DEVELOPMENT PART 2

# TITLE: CREATE CHATBOT USING PYTHON

## PHASE 4:

In this technology you will continue building your project by selecting a machine learning algorithm, training the model, and evaluating its performance. Perform different analysis as needed. After performing the relevant activities create a document around it and share the same for assessment.

Building a chatbot using Python involves several steps, including data collection, preprocessing, selecting a machine learning algorithm, training the model, and evaluating its performance

## PROBLEM DEFINITION:

The project aims to develop a rule-based chatbot in Python, a conversational agent capable of engaging in text-based interactions with users. Chatbots have gained significant popularity in recent years, finding applications in customer support, virtual assistants, and more.

**OBJECTIVES:**

• Implement a user-friendly interface that accepts text input from users and displays responses from the chatbot.

• Define a set of rules and responses to allow the chatbot to understand and generate appropriate replies to user inputs.

• Incorporate randomization to make the chatbot's responses less predictable and more engaging.

• Enable a termination condition, allowing users to exit the chatbot conversation when desired.

**DATASET:** https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot

## DATA PREPROCESSING:

Preprocess the dataset. This includes text cleaning, tokenization, removing stop words, and handling special characters.

**PROGRAM:**

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string
dataset_file = "chatbotdataset.csv"
data = pd.read_csv(dataset_file)
print(df.head())
print(df.isnull().sum())
df.fillna(df.mean(), inplace=True)
df.to_csv('preprocessed_dataset.csv', index=False)
df.fillna(df.mean(), inplace=True)
df.to_csv('preprocessed_dataset.csv', index=False)
def preprocess_text(text):
 text = text.lower()
 text = text.translate(str.maketrans('', '', string.punctuation))
 tokens = word_tokenize(text)
 stop_words = set(stopwords.words('english'))
 tokens = [word for word in tokens if word not in stop_words]
 cleaned_text = " ".join(tokens)
 return cleaned_text
data['text'] = data['text'].apply(preprocess_text)
```

The output will include the first few rows of the original dataset, the number of missing values in each column, and the first few rows of the preprocessed dataset. This program cleans the text, tokenizes it, removes stopwords, and saves the preprocessed dataset to a new CSV file named "preprocessed_dataset.csv."

```
df
```

| | hi, how are you doing? | i'm fine. how about yourself? |
|---|---|---|
| 0 | i'm fine. how about yourself? | i'm pretty good. thanks for asking. |
| 1 | i'm pretty good. thanks for asking. | no problem. so how have you been? |
| 2 | no problem. so how have you been? | i've been great. what about you? |
| 3 | i've been great. what about you? | i've been good. i'm in school right now. |
| 4 | i've been good. i'm in school right now. | what school do you go to? |
| ... | ... | ... |
| 3719 | that's a good question. maybe it's not old age. | are you right-handed? |
| 3720 | are you right-handed? | yes. all my life. |
| 3721 | yes. all my life. | you're wearing out your right hand. stop using... |
| 3722 | you're wearing out your right hand. stop using... | but i do all my writing with my right hand. |

## MACHINE LEARNING ALGORITHM:

### RULE BASED APPROACH:

A rule-based algorithm is an approach to creating a chatbot where the bot's responses are determined by a set of predefined rules and conditions. These rules can be based on pattern matching, regular expressions, or simple decision trees. Rule-based chatbots are effective for handling specific commands and straightforward interactions.

- Define a set of rules where each rule is a regular expression pattern and a corresponding response.
- The ‘respond_to_user_input’ function searches for a matching pattern in the user's input and returns the associated response if found.
- If no matching rule is found, the chatbot responds with a default "I don't understand" message.

The rule-based chatbot is suitable for handling specific commands or questions with predefined responses. Rule-based systems are straightforward to implement and can be highly effective in scenarios where interactions are predictable and rule-driven. However, they are not suitable for handling more complex or open-ended conversations.

## MODEL TRAINING:

Creating a rule-based chatbot does not involve traditional model training like machine learning models. Instead, it involves defining rules, patterns, or decision trees that determine the chatbot's responses based on user input. Here is a step-by-step guide to creating a rule-based chatbot in Python:

**1.Define Rules and Responses:**

Determine the rules and patterns that your chatbot will recognize. For example, you can create rules for greeting, asking questions, and other common interactions. Define responses for each rule.

**2.Implement a Rule-Based Chatbot:**

Create a Python script to implement your rule-based chatbot. Use regular expressions or simple if-else conditions to match user input to the defined rules.

**3.Interactive Chat Loop:**

Create an interactive chat loop where the chatbot continuously takes user input and responds based on the rules. The loop can continue until the user decides to exit.

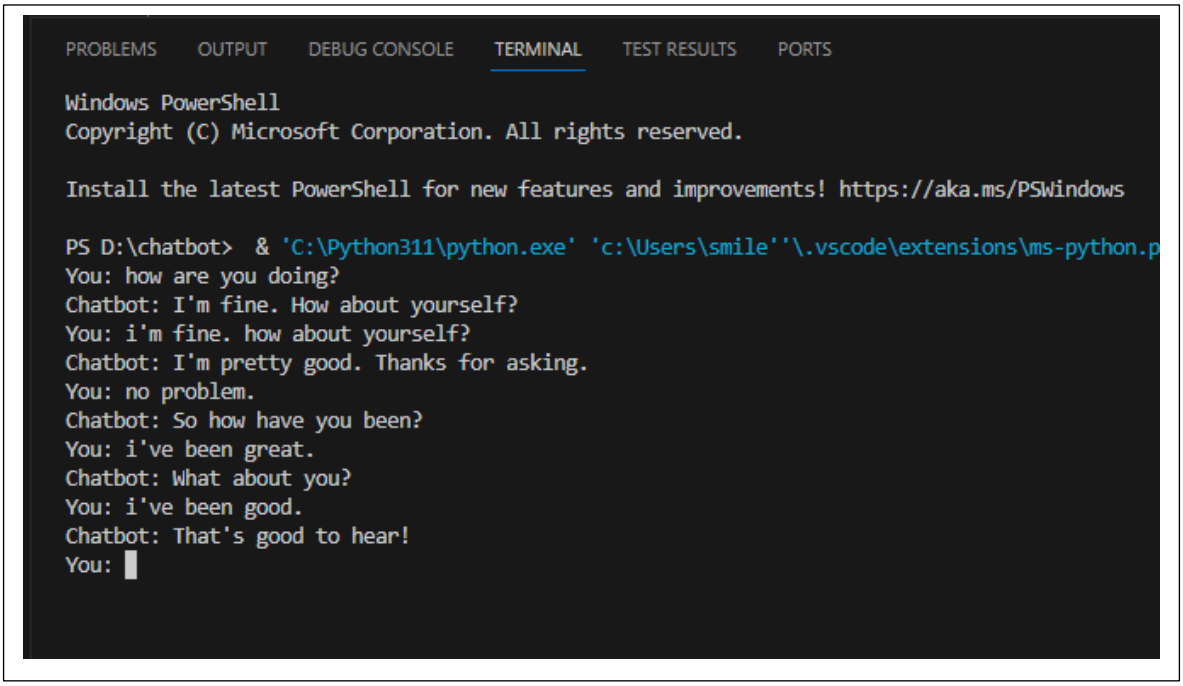
## CODE:

Creating a fully functional chatbot in Python is a comprehensive project that involves various components, including natural language processing, machine learning, and possibly integration with messaging platforms. A rule-based chatbot that responds to a few predefined commands. Here is a simple Python code for a rule-based chatbot:

```
import random
import csv

dataset_file = "chatbotdataset.csv"

responses = {}

# Open the CSV file
with open(dataset_file, mode='r') as file:
    csv_reader = csv.reader(file, delimiter=' ')  # Change delimiter to
match your CSV file
import random
import csv

dataset_file = "chatbotdataset.csv"

responses = {}

# Open the CSV file
with open(dataset_file, mode='r') as file:
    csv_reader = csv.reader(file, delimiter=' ')  # Change delimiter to
match your CSV file

    # Initialize an empty dictionary to store the responses
    responses = {}

    for row in csv_reader:
        if len(row) == 2:
            command, response = row[0], row[1]
            responses[command] = response

def get_response(user_input):
    user_input = user_input.lower()  # Convert to lowercase for case-
insensitivity
    for command, response in responses.items():
        if command in user_input:
            return response
    return "I don't understand. Please rephrase your question."
```

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

    if user_input.lower() == 'exit':
        print("Bot: Goodbye!")
        break

    response = get_response(user_input)
    print("Bot:", response)
```

# OUTPUT:



# DEPLOYMENT:

Deploying a rule-based chatbot in Python typically involves making it accessible to users via a web interface or messaging platform.

**Web Deployment:**

- Host a Web Server: Need of a web server to host chatbot. Common choices include Apache, Nginx, or cloud platforms like AWS, Google Cloud.

- Frontend: Creating a web interface where users can interact with the chatbot. We can use HTML, CSS, and JavaScript for this purpose. Embed a chat interface on your website.

- Backend: Host the Python script for your rule-based chatbot on the web server. Ensure that the web server is configured to execute Python scripts (e.g., using CGI or a web framework like Flask or Django).

- Integration: Integrate the frontend and backend to facilitate user interactions. Use AJAX or WebSocket for real-time communication with the chatbot backend.

- Domain and SSL: We have a custom domain, so we configure it to point to your web server. Additionally, consider enabling SSL (HTTPS) for secure communication.

- Security: Implement security measures on your web server to protect against attacks. Ensure that user input is properly validated and sanitized to prevent vulnerabilities like cross-site scripting (XSS).

## CONCLUSION:

In conclusion, creating a rule-based chatbot in Python is a practical and straightforward approach for developing chatbots with predefined responses to specific commands or keywords. Rule-based chatbots are effective for handling simple interactions and can serve as a valuable tool in scenarios where user inputs are well-defined. The process typically involves defining a set of rules and their corresponding responses, developing a chatbot logic, and deploying it through a web interface or messaging platform.

However, it is important to recognize that rule-based chatbots have limitations. They lack the ability to learn from user interactions and adapt to new or complex queries. They are constrained by the predefined rules and can struggle to handle open-ended conversations. To overcome these limitations and achieve more sophisticated chatbot capabilities, a transition to machine learning-based approaches like natural language processing (NLP) and deep learning may be necessary. Such approaches can enable chatbots to understand context, improve responses, and handle a broader range of user inputs.

In summary, rule-based chatbots are a great starting point for simple automation of responses to specific commands. They provide a foundation for chatbot development and can be a steppingstone toward more advanced, data-driven chatbots. Depending on the project's complexity and goals, it is essential to choose the most suitable approach, whether it is rule-based or machine learning-based, to deliver an optimal user experience.