Name: **Harsh Pramod Padyal**

Roll No : **40**

Div: **D20B**

Experiment 02

AIM: To build a Cognitive text based application to understand context for a Customer service application/ Insurance / Healthcare Application / Smarter Cities / Government etc.

Theory:

What is a Cognitive Text-Based Application?

A **cognitive text-based application** is an intelligent software system designed to:

- Understand human language (text or speech),
- Process and analyze user input using AI/NLP techniques,
- Generate appropriate responses based on context.

These applications mimic **human-like thinking** by incorporating **AI, Machine Learning (ML), and Cognitive Computing** principles to simulate decision-making and learning from data.

Understanding Cognitive Computing

Cognitive computing is the backbone of cognitive applications. It aims to replicate human cognitive functions like:

- **Language understanding** – Interpreting sentences and context.
- **Learning from interactions** – Adapting and improving over time.
- **Decision making** – Selecting the most relevant response or action.

Cognitive applications do not just respond based on fixed rules but can evolve with data and feedback.

Natural Language Processing (NLP)

To make sense of user input, cognitive applications rely on **Natural Language Processing** — a subfield of AI that helps machines interpret human language.

Key NLP techniques used:

1. **Tokenization** – Breaking text into words or phrases (tokens). Example: `"What's for dinner?"` → `["What", "'s", "for", "dinner", "?"]`
2. **Stopword Removal** – Removing common words (like "is", "the") that don't add much meaning.
3. **Lemmatization** – Converting words to their root form. Example: `"running", "ran"` → `"run"`
4. **Intent Recognition (via Keyword Matching)** – Identifying what the user wants based on key phrases.

In this experiment, **NLTK** and **spaCy** libraries are used to perform these tasks efficiently.

Application Design: Smart Kitchen Assistant

The application developed in this experiment is a **text-based kitchen assistant** that can understand and respond to cooking-related queries. It functions as a **rule-based chatbot** using keyword matching and NLP to determine user intent.

Core Components:

1. **Data Setup**: A list of predefined user queries (keywords) and appropriate responses.

```
queries_and_responses = [
    ("what's for dinner", "I can suggest a quick pasta dish..."),
    ("recipe for pasta", "For a basic pasta, you'll need..."),
    ...
]
```

2. **Text Preprocessing**: Using **nltk** and **spaCy** to tokenize and clean user input.
3. **Keyword Matching**: Matching user input to known queries using word comparison.
4. **Response Generation**: Selecting a response based on the matched intent or using a default fallback message.
5. **User Interaction Loop**: A continuous loop that accepts user input, processes it, and provides a response — simulating a real conversation.

Libraries Used:

- **spaCy**: For lemmatization, entity recognition, and language modeling.
- **NLTK**: For tokenization and stopword filtering.

Flow of Execution

1. User types a query like: `"Can you suggest a dinner recipe?"`
2. The system tokenizes and cleans the input.
3. It checks for keyword matches (e.g., "dinner", "recipe").
4. A matching response is fetched from the predefined list.
5. The assistant responds: `"I can suggest a quick pasta dish or a healthy salad..."`

If no match is found, it replies with a **default message** like: `"Hmm, I'm not sure I understood that. Can you rephrase?"`

Real-World Applications

Cognitive text-based systems can be used across many sectors:

- **Customer Support**: Automating FAQs, ticket generation.
- **Healthcare**: Symptom checkers, medication reminders.
- **Insurance**: Policy information, claim status updates.
- **Smart Cities**: Complaint redressal, utility updates.
- **Government Services**: Providing access to schemes, information on public services.

These applications:

- Save time,
- Improve accessibility,
- Provide round-the-clock assistance,
- Reduce load on human agents.

Install required libraries and download NLTK data

```
# Install spaCy (for advanced NLP features like lemmatization and stopword removal)
!pip install spacy --quiet
# Download the English language model for spaCy
!python -m spacy download en_core_web_sm --quiet
```

Import NLTK and download necessary resources

```
import nltk
nltk.download('punkt') # For tokenization
nltk.download('stopwords') # For stopword list
nltk.download('punkt_tab') # Download punkt_tab for tokenization
```

Import libraries and load spaCy model

```
import spacy
# Load the pre-trained English language model
nlp = spacy.load("en_core_web_sm")
```

Define sample queries and responses for a Smart Kitchen Assistant

This list contains common user queries (keywords) and the assistant's predefined responses. The keywords are simple phrases that the chatbot will look for in the user's input.

```
queries_and_responses = [
    ("what's for dinner", "I can suggest a quick pasta dish or a healthy salad. What are you in the mood for?"),
    ("recipe for pasta", "For a basic pasta, you'll need pasta, tomato sauce, garlic, and olive oil. Would you like
    ("how to chop onions", "To chop onions without tears, cut off the top, slice in half, peel, make horizontal cuts
    ("check pantry", "Your pantry has rice, lentils, canned tomatoes, and some spices. What are you looking for?"),
    ("shopping list", "I can add milk, eggs, and bread to your shopping list. Anything else?"),
    ("set timer for 10 minutes", "Timer set for 10 minutes! I'll let you know when it's done."),
    ("convert cups to ml", "1 cup is approximately 240 milliliters. What conversion do you need?"),
    ("clean the oven", "To clean the oven, spray with oven cleaner, let it sit for a few hours, then wipe clean. Rem
    ("leftovers in fridge", "You have leftover lasagna from yesterday and some stir-fry from two days ago. Should I
    ("healthy snack ideas", "How about an apple with peanut butter, or a handful of almonds?"),
]
```

Default responses for general interactions

```
default_responses = {
    "greeting": "Hi there! I'm your Smart Kitchen Assistant. How can I help you in the kitchen today?",
    "farewell": "Goodbye! Enjoy your cooking!",
    "default": "Hmm, I'm not sure I understood that. Can you rephrase or ask about cooking, recipes, or kitchen task
    "loading": "Just a moment, thinking...", # Added for a more conversational feel during processing
}
```

Function to classify user queries and generate responses

This function takes the user's input and tries to match it to a predefined query. It uses NLTK for tokenization and checks for keyword intersections.

```
def classify_query(user_query):
    # Convert the user query to lowercase for case-insensitive matching
    user_query_lower = user_query.lower()
    # Tokenize the user query into individual words
    user_tokens = set(nltk.word_tokenize(user_query_lower))
```

```
    # Check for greeting keywords
    if user_tokens.intersection({"hi", "hello", "hey", "hola"}):
        return "greeting"
    # Check for farewell keywords
    elif user_tokens.intersection({"bye", "goodbye", "see ya", "farewell"}):
        return "farewell"

    # Iterate through the predefined queries and responses
    for keywords_str, response in queries_and_responses:
        # Tokenize the keywords for the current predefined query
        keyword_tokens = set(nltk.word_tokenize(keywords_str.lower()))
        # Check if there's any overlap (intersection) between user tokens and keyword tokens
        if user_tokens.intersection(keyword_tokens):
            return response # Return the corresponding predefined response

    # If no specific match is found, return "default"
    return "default"
```

Main chatbot loop for interaction

```
def chatbot():
    print("Smart Kitchen Assistant: " + default_responses["greeting"])
    while True:
        user_query = input("You: ") # Get input from the user

        # Simulate a brief processing time for a more natural interaction
        print("Smart Kitchen Assistant:", default_responses["loading"])

        # Classify the user's query
        response_type = classify_query(user_query)

        # Respond based on the classified query type
        if response_type == "greeting":
            print("Smart Kitchen Assistant:", default_responses["greeting"])
        elif response_type == "farewell":
            print("Smart Kitchen Assistant:", default_responses["farewell"])
            break # Exit the loop if the user says goodbye
        elif response_type != "default":
            # If a specific response was found, print it
            print("Smart Kitchen Assistant:", response_type)
        else:
            # If no specific match, print the default "didn't understand" message
            print("Smart Kitchen Assistant:", default_responses["default"])
```

Run the chatbot

```
# This ensures the chatbot starts when the script is executed
if __name__ == "__main__":
    chatbot()
```

```
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

```
Smart Kitchen Assistant: Hi there! I'm your Smart Kitchen Assistant. How can I help you in the kitchen today?
You: Hi
Smart Kitchen Assistant: Just a moment, thinking...
Smart Kitchen Assistant: Hi there! I'm your Smart Kitchen Assistant. How can I help you in the kitchen today?
You: What's for dinner tonight?
Smart Kitchen Assistant: Just a moment, thinking...
Smart Kitchen Assistant: I can suggest a quick pasta dish or a healthy salad. What are you in the mood for?
You: How do I chop onions?
Smart Kitchen Assistant: Just a moment, thinking...
Smart Kitchen Assistant: To chop onions without tears, cut off the top, slice in half, peel, make horizontal cut
You: What's in the pantry?
Smart Kitchen Assistant: Just a moment, thinking...
Smart Kitchen Assistant: I can suggest a quick pasta dish or a healthy salad. What are you in the mood for?
You: Set a timer for 10 minutes
Smart Kitchen Assistant: Just a moment, thinking...
Smart Kitchen Assistant: I can suggest a quick pasta dish or a healthy salad. What are you in the mood for?
You: How to clean the oven?
Smart Kitchen Assistant: Just a moment, thinking...
Smart Kitchen Assistant: To chop onions without tears, cut off the top, slice in half, peel, make horizontal cut
You: Healthy snack ideas
Smart Kitchen Assistant: Just a moment, thinking...
Smart Kitchen Assistant: How about an apple with peanut butter, or a handful of almonds?
You: Bye
Smart Kitchen Assistant: Just a moment, thinking...
Smart Kitchen Assistant: Goodbye! Enjoy your cooking!
```

Conclusion

This experiment successfully demonstrates the implementation of a **basic cognitive text-based assistant** using Python and NLP libraries. Though rule-based in design, it showcases how computers can simulate intelligent conversation, paving the way for more advanced AI-powered virtual assistants.

By applying concepts of **cognitive computing, NLP, and intent recognition**, this chatbot mimics human-like understanding and enhances interaction between users and machines in practical domains like **kitchen assistance**, and can be extended to many real-world applications.
