Ex.No: **Date:**

Prompt-Based AI Application for Personal Needs

Aim:

The aim of this experiment is to develop a customizable, prompt-based application that leverages large language models (LLMs) to assist users in solving personal or professional problems, fostering creativity and practical problem-solving skills. The application will allow users to input their specific needs, generate tailored responses, and refine outputs iteratively for optimal results.

Procedure:

1. Define User Requirements

- Identify the user's personal or professional needs (e.g., content creation, coding help, study assistance, brainstorming).
- Design prompts that guide the AI to generate useful responses.

2. Select AI Tools

- Choose an LLM (e.g., OpenAI's GPT, Hugging Face's models, or Google's Gemini).
- Decide whether to use cloud-based APIs or locally hosted models.

3. Develop the Application

- Write Python code to:
    - Take user input (custom prompts).
    - Query the AI model(s).
    - Display and refine responses.
- Implement error handling and logging.

4. Test and Refine

- Run the application with different prompts.
- Compare outputs and adjust prompts for better results.
- Add features like response history or multi-model comparison.

5. Deploy (Optional)

- Convert the script into a web app (using Flask/Streamlit) or a CLI tool.

Program (Python Code)

Python code:

```
import openai
import logging

# Configure logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

# Initialize OpenAI API (Replace with your API key)
openai.api_key = "your_openai_api_key"

def get_ai_response(prompt, model="gpt-3.5-turbo", max_tokens=300, temperature=0.7):
    """Get AI-generated response based on user prompt."""
    try:
        response = openai.ChatCompletion.create(
            model=model,
            messages=[{"role": "user", "content": prompt}],
            max_tokens=max_tokens,
            temperature=temperature
        )
        return response.choices[0].message['content'].strip()
    except Exception as e:
        logger.error(f"Error in AI request: {e}")
        return "Error: Failed to generate response."

def main():
    print("=== AI Prompt-Based Assistant ===")
    print("Enter your prompt (e.g., 'Help me write a resume', 'Explain quantum computing simply').")
```

```
    print("Type 'quit' to exit.\n")

    while True:
        user_prompt = input("\nYour Prompt: ")

        if user_prompt.lower() in ["quit", "exit"]:
            print("Exiting the application. Goodbye!")
            break

        if not user_prompt.strip():
            print("Please enter a valid prompt.")
            continue

        print("\nGenerating response...")
        ai_response = get_ai_response(user_prompt)
        print("\nAI Response:\n", ai_response)

if __name__ == "__main__":
    main()
```

Output Example:

User Input:

"Help me write a professional summary for a data scientist resume."

AI Response:

"Results-driven Data Scientist with 5+ years of experience in machine learning, statistical analysis, and big data processing. Proficient in Python, SQL, and TensorFlow, with a strong track record of developing predictive models that improve business decision-making. Passionate about transforming raw data into actionable insights and collaborating with cross-functional teams to drive innovation."

User Refinement:

"Make it shorter and focus on Python and deep learning expertise."

Refined AI Response:

"Data Scientist specializing in Python and deep learning, with expertise in building neural networks for predictive analytics. Strong background in TensorFlow and PyTorch, delivering scalable AI solutions."

Result:

- The application successfully generates tailored responses based on user prompts.
- Users can iteratively refine prompts to get better results.
- The modular code allows integration with other AI models (e.g., Hugging Face, Claude).
- Future enhancements could include multi-model comparison, sentiment analysis, or a GUI.