SRM Institute of Science and Technology

Department of Computer Applications

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Circular – 2020-21

MCA GAI 3rd semester

Building Conversational AI for Human Resources (PGI20D18J)

Lab Manual

Lab 1: Implement a tokenization function to split input text into tokens.

Title: Tokenization Function Implementation

Aim: To implement a basic tokenization function in Python that splits an input text into individual words and punctuation marks (tokens).

Procedure:

1. Define a Python function, `tokenize_text`, that accepts a single string argument, `text`.
2. Inside the function, use regular expressions to find all sequences of word characters (alphanumeric and underscore) and individual punctuation marks.
3. Compile the regular expression for efficiency if the function is called multiple times.
4. Return a list of the extracted tokens.

Source Code:

```
import re

def tokenize_text(text):
    """
    Splits the input text into tokens (words and punctuation).

    Args:
        text (str): The input string to be tokenized.

    Returns:
        list: A list of tokens.
    """
    # Regex to match words (alphanumeric + underscore) or punctuation
    # \w+ matches one or more word characters
    # \S matches any non-whitespace character (used for punctuation)
    # The | acts as an OR, so it matches either a word or a single punctuation
mark.
    token_pattern = re.compile(r'\w+|[^\w\s]')
    tokens = token_pattern.findall(text)
    return tokens

# Example usage:
if __name__ == "__main__":
    sample_text_1 = "Hello, world! How are you today?"
    tokens_1 = tokenize_text(sample_text_1)
    print(f"Text: '{sample_text_1}'")
    print(f"Tokens: {tokens_1}")
```

```
    sample_text_2 = "User's leave request. (Approved)"
    tokens_2 = tokenize_text(sample_text_2)
    print(f"\nText: '{sample_text_2}'")
    print(f"Tokens: {tokens_2}")
```

Input:

```
"Hello, world! How are you today?"
"User's leave request. (Approved)"
```

Expected Output:

```
Text: 'Hello, world! How are you today?'
Tokens: ['Hello', ',', 'world', '!', 'How', 'are', 'you', 'today', '?']

Text: 'User's leave request. (Approved)'
Tokens: ['User', "'", 's', 'leave', 'request', '.', '(', 'Approved', ')']
```

Lab 2: Use an NLP library to perform part-of-speech tagging on sample sentences.

Title: Part-of-Speech (POS) Tagging with an NLP Library

Aim: To utilize an existing Natural Language Processing (NLP) library (e.g., NLTK or spaCy) to perform part-of-speech tagging on a set of sample sentences, identifying the grammatical category of each word.

Procedure:

1. **Installation:** Ensure the chosen NLP library (e.g., NLTK) and its necessary data (e.g., `punkt` tokenizer, `averaged_perceptron_tagger`) are installed and downloaded.
2. **Import:** Import the required modules from the library.
3. **Tokenization:** Tokenize the sample sentences into words.
4. **POS Tagging:** Apply the POS tagging function from the library to the tokens.
5. **Display:** Print each word along with its assigned POS tag.

Source Code:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag

# Download necessary NLTK data (run once)
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')
try:
    nltk.data.find('taggers/averaged_perceptron_tagger')
except nltk.downloader.DownloadError:
    nltk.download('averaged_perceptron_tagger')

def perform_pos_tagging(sentence):
    """
    Performs part-of-speech tagging on a given sentence using NLTK.

    Args:
        sentence (str): The input sentence.

    Returns:
        list: A list of (word, POS_tag) tuples.
    """
    # Tokenize the sentence into words
    words = word_tokenize(sentence)
    # Perform POS tagging
    tagged_words = pos_tag(words)
    return tagged_words

# Example usage:
if __name__ == "__main__":
    sample_sentence_1 = "The quick brown fox jumps over the lazy dog."
    pos_tags_1 = perform_pos_tagging(sample_sentence_1)
    print(f"Sentence: '{sample_sentence_1}'")
    print("POS Tags:")
    for word, tag in pos_tags_1:
        print(f"  {word}: {tag}")
```

```
    sample_sentence_2 = "John wants to request leave for two days."
    pos_tags_2 = perform_pos_tagging(sample_sentence_2)
    print(f"\nSentence: '{sample_sentence_2}'")
    print("POS Tags:")
    for word, tag in pos_tags_2:
        print(f"  {word}: {tag}")
```

Input:

```
"The quick brown fox jumps over the lazy dog."
"John wants to request leave for two days."
```

Expected Output:

```
Sentence: 'The quick brown fox jumps over the lazy dog.'
POS Tags:
  The: DT
  quick: JJ
  brown: JJ
  fox: NN
  jumps: VBZ
  over: IN
  the: DT
  lazy: JJ
  dog: NN
  .: .

Sentence: 'John wants to request leave for two days.'
POS Tags:
  John: NNP
  wants: VBZ
  to: TO
  request: VB
  leave: NN
  for: IN
  two: CD
  days: NNS
  .: .
```

Lab 3: Designing Conversational Flows: Define common HR scenarios (e.g., employee onboarding, leave request) and outline the conversation flow for each scenario.

Title: Designing Conversational Flows for HR Scenarios

Aim: To design and outline detailed conversational flows for common Human Resources (HR) scenarios, mapping out user interactions and bot responses to achieve specific HR tasks.

Procedure:

1. **Scenario Identification:** Choose at least two common HR scenarios (e.g., "Employee Onboarding Information" and "Leave Request Submission").
2. **User Intent Definition:** For each scenario, clearly define the primary user intent that initiates the conversation.
3. **Flow Mapping:** For each scenario, outline the step-by-step conversation flow, including:
    - Initial user utterance.
    - Bot's initial response and information gathering.
    - Subsequent user inputs (e.g., providing details, confirming information).
    - Bot's responses, clarifications, and confirmations.
    - Handling of missing or invalid information (simple error handling).
    - Confirmation of task completion.
4. **Edge Cases:** Briefly consider potential deviations or alternative paths within the conversation.

Source Code: (Conceptual Design - Textual Outline)

```
### Scenario 1: Employee Onboarding Information

**User Intent:** To get information about the onboarding process or specific
onboarding documents.

**Conversation Flow:**

1.  **User:** "Hi, I'm a new employee. Can you tell me about the onboarding
process?"
2.  **Bot:** "Welcome! I can help you with onboarding information. What specific
information are you looking for? For example, are you interested in required
documents, first-day instructions, or benefits enrollment?"
3.  **User:** "What documents do I need to submit?"
4.  **Bot:** "For onboarding, you'll typically need to submit your Aadhar Card,
PAN Card, educational certificates, previous employment documents, and bank
account details. Would you like a checklist of these documents?"
5.  **User:** "Yes, please provide the checklist."
6.  **Bot:** "Certainly. I've sent a link to the complete onboarding document
checklist to your registered email address. Is there anything else I can assist
you with regarding onboarding?"
7.  **User:** "No, that's all for now. Thank you!"
8.  **Bot:** "You're welcome! Feel free to ask if you have more questions."

---

### Scenario 2: Leave Request Submission

**User Intent:** To submit a leave request.
```

```
**Conversation Flow:**

1.  **User:** "I want to apply for leave."
2.  **Bot:** "Okay, I can help you with that. What type of leave are you
applying for (e.g., casual leave, sick leave, annual leave)?"
3.  **User:** "Sick leave."
4.  **Bot:** "Understood. What are the start and end dates for your sick leave?"
5.  **User:** "From tomorrow, May 23rd, to May 25th."
6.  **Bot:** "So, you're requesting sick leave from May 23rd to May 25th. Is
that correct? (3 days total)"
7.  **User:** "Yes, that's correct."
8.  **Bot:** "Please provide a brief reason for your sick leave."
9.  **User:** "Feeling unwell, need rest."
10. **Bot:** "Thank you. Your sick leave request from May 23rd to May 25th with
the reason 'Feeling unwell, need rest' has been submitted for approval. You will
receive a notification once it's approved. Is there anything else?"
11. **User:** "No, thanks."
12. **Bot:** "You're welcome!"

**Edge Cases/Considerations:**
* User provides invalid dates (e.g., past dates, end date before start date).
* User asks about leave balance before applying.
* User changes mind about leave type or dates mid-conversation.
```

Input: N/A (Conceptual Design)

Expected Output: Detailed, step-by-step textual outlines of conversational flows for the chosen HR scenarios, including user utterances and bot responses.

Lab 4: Construct dialogue trees representing various conversation paths based on user inputs and system responses.

Title: Constructing Dialogue Trees

Aim: To construct visual or textual dialogue trees that represent different conversation paths, illustrating how the conversational AI system responds to various user inputs and guides the conversation.

Procedure:

1. **Scenario Selection:** Choose a simple HR scenario from Lab 3 (e.g., "Leave Request Inquiry").
2. **Identify Intents and Entities:** Determine the main intents (e.g., `request_leave`, `check_leave_balance`) and entities (e.g., `leave_type`, `start_date`, `end_date`) relevant to the scenario.
3. **Map Paths:** Begin with a root node (initial user utterance) and branch out based on possible user responses or system prompts.
4. **Nodes and Edges:**
    - **Nodes:** Represent a state in the conversation (e.g., bot prompt, user input, action taken).
    - **Edges:** Represent the transition between states based on user input or system logic.
5. **Include Variations:** Show how the tree branches for different user inputs (e.g., providing all info at once, providing info incrementally, asking for clarification).
6. **Termination:** Clearly mark the end points of successful or unsuccessful conversation paths.

Source Code: (Conceptual Design - Textual Representation of a Dialogue Tree)

```
### Dialogue Tree for "Leave Request Inquiry"

**Root Node (User Intent: `request_leave`)**

* **User:** "I want to apply for leave."
    * **Bot Prompt:** "Okay, I can help you with that. What type of leave are
you applying for (e.g., casual leave, sick leave, annual leave)?"

    * **Path 1: User provides leave type**
        * **User:** "Sick leave."
        * **Bot Prompt:** "Understood. What are the start and end dates for your
sick leave?"

        * **Path 1.1: User provides valid dates**
            * **User:** "From May 23rd to May 25th."
            * **Bot Confirmation:** "So, you're requesting sick leave from May
23rd to May 25th. Is that correct?"
            * **User:** "Yes."
            * **Bot Prompt:** "Please provide a brief reason for your sick
leave."
            * **User:** "Feeling unwell."
            * **Bot Action/Confirmation:** "Your sick leave request has been
submitted for approval. You will receive a notification. Is there anything
else?"
            * **User:** "No, thanks." -> **End Conversation**

        * **Path 1.2: User provides invalid dates / asks for help**
            * **User:** "I'm not sure about the dates."
```

```
            * **Bot Clarification:** "Please provide the exact start and end
dates. For example, 'May 23rd to May 25th'."
            * **(Loop back to "Bot Prompt: What are the start and end
dates...")**

    * **Path 2: User asks about leave balance first**
        * **User:** "What's my leave balance?"
        * **Bot Action:** "Checking your leave balance... You have 5 casual
leaves and 3 sick leaves remaining. Would you still like to apply for leave?"
        * **User:** "Yes, sick leave."
        * **(Merge back to "Bot Prompt: Understood. What are the start and end
dates...")**

    * **Path 3: User provides all information at once (advanced)**
        * **User:** "I want to apply for sick leave from May 23rd to May 25th
because I'm unwell."
        * **Bot Confirmation:** "So, you're requesting sick leave from May 23rd
to May 25th with the reason 'I'm unwell'. Is that correct?"
        * **User:** "Yes."
        * **Bot Action/Confirmation:** "Your sick leave request has been
submitted for approval. You will receive a notification. Is there anything
else?"
        * **User:** "No, thanks." -> **End Conversation**

    * **Path 4: User provides irrelevant input**
        * **User:** "Tell me a joke."
        * **Bot Error Handling:** "I'm sorry, I can only assist with HR-related
queries. Would you like to apply for leave or check your leave balance?"
        * **(Loop back to initial bot prompt or offer options)**
```

Input: N/A (Conceptual Design)

Expected Output: A clear, structured textual representation of a dialogue tree for a chosen HR scenario, illustrating various conversation paths based on user inputs and system responses.

Lab 5: Implement error handling strategies to handle user misunderstandings or unexpected inputs.

Title: Implementing Conversational Error Handling

Aim: To implement robust error handling strategies within a conversational AI system to gracefully manage user misunderstandings, ambiguous inputs, or unexpected queries, guiding the user back to a productive conversation.

Procedure:

1. **Identify Error Types:** Categorize common errors (e.g., unrecognized intent, missing entity, invalid format, out-of-scope query).
2. **Fallback Intent:** Implement a "fallback" or "no-match" intent that triggers when the user's input cannot be understood by any defined intent.
3. **Clarification Prompts:** For missing entities or ambiguous inputs, design specific clarification prompts to ask the user for the required information.
4. **Reprompting:** If the user repeatedly provides invalid input, implement a mechanism to reprompt them or offer alternative actions.
5. **Out-of-Scope Handling:** For queries completely outside the bot's domain, provide a polite message indicating the bot's limitations and guiding them to relevant topics.
6. **Example Implementation:** Demonstrate these strategies within a simple conversational flow.

Source Code: (Python Pseudo-code / Logic Outline)

```
def process_user_input(user_input, current_context):
    """
    Simulates processing user input and applies error handling.

    Args:
        user_input (str): The text input from the user.
        current_context (dict): The current state/context of the conversation.

    Returns:
        tuple: (bot_response, new_context)
    """
    # 1. Intent Recognition (Simplified for demonstration)
    recognized_intent = recognize_intent(user_input)
    extracted_entities = extract_entities(user_input)

    if recognized_intent == "request_leave":
        # Check for required entities
        if not extracted_entities.get("leave_type"):
            return "What type of leave are you applying for?", current_context
        if not extracted_entities.get("start_date") or not
extracted_entities.get("end_date"):
            return "Please provide the start and end dates for your leave.",
current_context
        # Simulate successful processing
        return "Your leave request has been submitted.", {} # Reset context

    elif recognized_intent == "check_leave_balance":
        return "Checking your leave balance... You have 5 casual and 3 sick
leaves.", {}

    # 2. Error Handling - Unrecognized Intent (Fallback)
```

```
    elif recognized_intent == "unrecognized_intent":
        # Check if it's a repeated misunderstanding
        if current_context.get("misunderstanding_count", 0) >= 2:
            current_context["misunderstanding_count"] = 0 # Reset count
            return "I'm having trouble understanding. Could you please rephrase
your request or tell me if you want to 'apply for leave' or 'check leave
balance'?", current_context
        else:
            current_context["misunderstanding_count"] =
current_context.get("misunderstanding_count", 0) + 1
            return "I'm sorry, I didn't understand that. Could you please try
again?", current_context

    # 3. Error Handling - Out of Scope
    elif recognized_intent == "out_of_scope":
        return "I can only assist with HR-related queries like leave requests or
onboarding information. How can I help you with HR today?", current_context

    # Default fallback
    return "I'm sorry, I couldn't process your request. Can you please be more
specific?", current_context

def recognize_intent(text):
    """Simulates intent recognition."""
    text_lower = text.lower()
    if "leave" in text_lower and ("apply" in text_lower or "request" in
text_lower):
        return "request_leave"
    elif "leave balance" in text_lower or "how much leave" in text_lower:
        return "check_leave_balance"
    elif "joke" in text_lower or "weather" in text_lower:
        return "out_of_scope"
    else:
        return "unrecognized_intent"

def extract_entities(text):
    """Simulates entity extraction."""
    entities = {}
    text_lower = text.lower()
    if "sick leave" in text_lower:
        entities["leave_type"] = "sick"
    if "casual leave" in text_lower:
        entities["leave_type"] = "casual"
    # Simple date extraction (for demo purposes)
    if "may 23rd" in text_lower:
        entities["start_date"] = "May 23rd"
    if "may 25th" in text_lower:
        entities["end_date"] = "May 25th"
    return entities

# Example Conversation Simulation:
if __name__ == "__main__":
    conversation_context = {}

    print("User: I want to apply for leave.")
    response, conversation_context = process_user_input("I want to apply for
leave.", conversation_context)
    print(f"Bot: {response}")

    print("\nUser: Just leave.") # Missing type
    response, conversation_context = process_user_input("Just leave.",
conversation_context)
    print(f"Bot: {response}")

    print("\nUser: Sick leave.") # Missing dates
    response, conversation_context = process_user_input("Sick leave.",
conversation_context)
```

```
    print(f"Bot: {response}")

    print("\nUser: From May 23rd to May 25th.") # Now complete
    response, conversation_context = process_user_input("From May 23rd to May
25th.", conversation_context)
    print(f"Bot: {response}")

    print("\nUser: Tell me a joke.") # Out of scope
    response, conversation_context = process_user_input("Tell me a joke.",
conversation_context)
    print(f"Bot: {response}")

    print("\nUser: abcdefg") # Unrecognized
    response, conversation_context = process_user_input("abcdefg",
conversation_context)
    print(f"Bot: {response}")

    print("\nUser: xyz") # Unrecognized (repeated)
    response, conversation_context = process_user_input("xyz",
conversation_context)
    print(f"Bot: {response}")

    print("\nUser: What is my leave balance?") # Recognized
    response, conversation_context = process_user_input("What is my leave
balance?", conversation_context)
    print(f"Bot: {response}")
```

Input:

```
User: I want to apply for leave.
User: Just leave.
User: Sick leave.
User: From May 23rd to May 25th.
User: Tell me a joke.
User: abcdefg
User: xyz
User: What is my leave balance?
```

Expected Output:

```
User: I want to apply for leave.
Bot: What type of leave are you applying for?

User: Just leave.
Bot: What type of leave are you applying for?

User: Sick leave.
Bot: Please provide the start and end dates for your leave.

User: From May 23rd to May 25th.
Bot: Your leave request has been submitted.

User: Tell me a joke.
Bot: I can only assist with HR-related queries like leave requests or onboarding
information. How can I help you with HR today?

User: abcdefg
Bot: I'm sorry, I didn't understand that. Could you please try again?

User: xyz
Bot: I'm having trouble understanding. Could you please rephrase your request or
tell me if you want to 'apply for leave' or 'check leave balance'?

User: What is my leave balance?
```

Bot: Checking your leave balance... You have 5 casual and 3 sick leaves.

Lab 6: Design and train a simple intent recognition model using a rule-based or machine learning approach.

Title: Simple Intent Recognition Model

Aim: To design and implement a basic intent recognition model that can classify user utterances into predefined HR-related intents using either a rule-based approach or a simple machine learning technique.

Procedure:

1. **Define Intents:** Identify a few core HR intents (e.g., `apply_leave`, `check_balance`, `onboarding_info`).
2. **Collect Training Data (if ML):** Create a small dataset of example utterances for each intent.
3. **Choose Approach:**
    - **Rule-based:** Define keywords or patterns for each intent.
    - **Machine Learning (e.g., Naive Bayes, SVM, or simple neural network):**
        - **Feature Extraction:** Convert text into numerical features (e.g., Bag-of-Words, TF-IDF).
        - **Model Training:** Train a classifier on the prepared data.
4. **Implementation:** Write code to implement the chosen approach.
5. **Testing:** Test the model with new, unseen utterances.

Source Code: (Python - Simple Rule-Based Approach for clarity)

```
def recognize_intent_rule_based(text):
    """
    Recognizes HR-related intents based on predefined rules (keywords).

    Args:
        text (str): The user's input utterance.

    Returns:
        str: The recognized intent or 'unrecognized'.
    """
    text_lower = text.lower()

    if "apply" in text_lower and ("leave" in text_lower or "vacation" in
text_lower):
        return "apply_leave"
    elif "check" in text_lower and ("balance" in text_lower or "leaves" in
text_lower):
        return "check_leave_balance"
    elif "onboarding" in text_lower or "new employee" in text_lower or "join" in
text_lower:
        return "onboarding_info"
    elif "hi" in text_lower or "hello" in text_lower:
        return "greet"
    elif "bye" in text_lower or "goodbye" in text_lower:
        return "farewell"
    else:
        return "unrecognized"

# Example usage:
if __name__ == "__main__":
    utterances = [
```

```
        "I want to apply for sick leave.",
        "How many leaves do I have?",
        "Tell me about new employee onboarding.",
        "Hello there!",
        "Goodbye for now.",
        "What is the weather today?",
        "Can I request a vacation?",
        "Check my leave balance.",
        "I need information about joining the company."
    ]

    print("--- Intent Recognition Results (Rule-Based) ---")
    for utterance in utterances:
        intent = recognize_intent_rule_based(utterance)
        print(f"Utterance: '{utterance}' -> Intent: '{intent}'")

    # --- Alternative: Simple ML Approach (Conceptual Outline) ---
    # This section is commented out as it requires more setup (libraries, data)
    # but provides an idea of an ML approach.

    # from sklearn.feature_extraction.text import CountVectorizer
    # from sklearn.naive_bayes import MultinomialNB
    # from sklearn.pipeline import Pipeline
    # from sklearn.model_selection import train_test_split
    # from sklearn.metrics import classification_report

    # # Sample Data for ML
    # ml_data = [
    #     ("I want to apply for leave", "apply_leave"),
    #     ("Can I request a vacation?", "apply_leave"),
    #     ("Submit a leave application", "apply_leave"),
    #     ("How many leaves do I have?", "check_balance"),
    #     ("What's my leave balance?", "check_balance"),
    #     ("Check my remaining leaves", "check_balance"),
    #     ("Tell me about onboarding", "onboarding_info"),
    #     ("New employee guide", "onboarding_info"),
    #     ("Information for joining", "onboarding_info"),
    #     ("Hi", "greet"),
    #     ("Hello", "greet"),
    #     ("Good morning", "greet"),
    #     ("Bye", "farewell"),
    #     ("See you later", "farewell")
    # ]

    # X = [item[0] for item in ml_data]
    # y = [item[1] for item in ml_data]

    # # Split data (optional for small datasets, but good practice)
    # X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    # # Create a pipeline: Vectorizer -> Classifier
    # text_clf = Pipeline([
    #     ('vect', CountVectorizer()), # Converts text to token counts
    #     ('clf', MultinomialNB()),   # Naive Bayes classifier
    # ])

    # # Train the model
    # # print("\n--- Training ML Model (Conceptual) ---")
    # # text_clf.fit(X_train, y_train)
    # # print("Model trained.")

    # # Predict and evaluate (conceptual)
    # # predictions = text_clf.predict(X_test)
    # # print("\n--- ML Model Evaluation (Conceptual) ---")
    # # print(classification_report(y_test, predictions))
```

```
    # # Example prediction with ML model (conceptual)
    # # new_utterance_ml = "I need to apply for annual leave."
    # # predicted_intent_ml = text_clf.predict([new_utterance_ml])[0]
    # # print(f"\nML Prediction: '{new_utterance_ml}' -> Intent:
'{predicted_intent_ml}'")
```

Input:

```
"I want to apply for sick leave."
"How many leaves do I have?"
"Tell me about new employee onboarding."
"Hello there!"
"Goodbye for now."
"What is the weather today?"
"Can I request a vacation?"
"Check my leave balance."
"I need information about joining the company."
```

Expected Output:

```
--- Intent Recognition Results (Rule-Based) ---
Utterance: 'I want to apply for sick leave.' -> Intent: 'apply_leave'
Utterance: 'How many leaves do I have?' -> Intent: 'check_leave_balance'
Utterance: 'Tell me about new employee onboarding.' -> Intent: 'onboarding_info'
Utterance: 'Hello there!' -> Intent: 'greet'
Utterance: 'Goodbye for now.' -> Intent: 'farewell'
Utterance: 'What is the weather today?' -> Intent: 'unrecognized'
Utterance: 'Can I request a vacation?' -> Intent: 'apply_leave'
Utterance: 'Check my leave balance.' -> Intent: 'check_leave_balance'
Utterance: 'I need information about joining the company.' -> Intent:
'onboarding_info'
```

(Note: If a Machine Learning approach is implemented, the output would include training/evaluation metrics and predictions based on the trained model.)

Lab 7: Access HR-related data sources (e.g., employee database, leave management system) using APIs or database queries.

Title: Accessing HR Data Sources

Aim: To simulate accessing HR-related data (e.g., employee information, leave records) from a data source using a mock API or direct database interaction (represented by a Python dictionary/list for simplicity).

Procedure:

1. **Define Mock Data:** Create a Python dictionary or list of dictionaries to represent a simplified HR database (e.g., `employees` with `id`, `name`, `department`, `leave_balance`; `leave_requests` with `employee_id`, `type`, `start_date`, `end_date`, `status`).
2. **Simulate API/DB Functions:** Write Python functions that simulate API calls or database queries to:
    - Retrieve employee details by ID or name.
    - Fetch leave balance for an employee.
    - Retrieve pending leave requests.
    - (Optional) Simulate updating a leave request status.
3. **Error Handling:** Include basic error handling for cases where data is not found.
4. **Demonstrate Access:** Call these functions with sample inputs and print the retrieved data.

Source Code: (Python - Mock Data and Access Functions)

```
# Mock HR Database
HR_DATABASE = {
    "employees": [
        {"id": "EMP001", "name": "Alice Smith", "department": "HR", "email":
"alice.s@example.com", "leave_balance": {"sick": 10, "casual": 15}},
        {"id": "EMP002", "name": "Bob Johnson", "department": "Engineering",
"email": "bob.j@example.com", "leave_balance": {"sick": 8, "casual": 12}},
        {"id": "EMP003", "name": "Charlie Brown", "department": "Sales",
"email": "charlie.b@example.com", "leave_balance": {"sick": 5, "casual": 10}}
    ],
    "leave_requests": [
        {"request_id": "LR001", "employee_id": "EMP001", "type": "sick",
"start_date": "2025-06-01", "end_date": "2025-06-03", "status": "pending",
"reason": "Fever"},
        {"request_id": "LR002", "employee_id": "EMP002", "type": "casual",
"start_date": "2025-07-10", "end_date": "2025-07-12", "status": "approved",
"reason": "Family event"},
        {"request_id": "LR003", "employee_id": "EMP001", "type": "casual",
"start_date": "2025-08-05", "end_date": "2025-08-05", "status": "pending",
"reason": "Personal work"}
    ]
}

def get_employee_details(employee_id=None, employee_name=None):
    """
    Simulates fetching employee details from the HR database.
    """
    for emp in HR_DATABASE["employees"]:
        if employee_id and emp["id"].lower() == employee_id.lower():
            return emp
```

```
        if employee_name and emp["name"].lower() == employee_name.lower():
            return emp
    return None

def get_leave_balance(employee_id):
    """
    Simulates fetching an employee's leave balance.
    """
    employee = get_employee_details(employee_id=employee_id)
    if employee:
        return employee["leave_balance"]
    return None

def get_pending_leave_requests(employee_id=None):
    """
    Simulates fetching pending leave requests.
    If employee_id is provided, filters by employee.
    """
    pending_requests = []
    for req in HR_DATABASE["leave_requests"]:
        if req["status"].lower() == "pending":
            if employee_id is None or req["employee_id"].lower() ==
employee_id.lower():
                pending_requests.append(req)
    return pending_requests

def submit_leave_request(employee_id, leave_type, start_date, end_date, reason):
    """
    Simulates submitting a new leave request.
    In a real system, this would add to a database.
    """
    new_request_id = f"LR{len(HR_DATABASE['leave_requests']) + 1:03d}"
    new_request = {
        "request_id": new_request_id,
        "employee_id": employee_id,
        "type": leave_type,
        "start_date": start_date,
        "end_date": end_date,
        "status": "pending",
        "reason": reason
    }
    HR_DATABASE["leave_requests"].append(new_request)
    return new_request

# Example usage:
if __name__ == "__main__":
    print("--- Accessing HR Data ---")

    # Get employee details by ID
    emp_id = "EMP001"
    alice_details = get_employee_details(employee_id=emp_id)
    print(f"\nDetails for {emp_id}: {alice_details}")

    # Get employee details by name
    emp_name = "Bob Johnson"
    bob_details = get_employee_details(employee_name=emp_name)
    print(f"\nDetails for {emp_name}: {bob_details}")

    # Get leave balance
    alice_leave_balance = get_leave_balance(employee_id="EMP001")
    print(f"\nAlice's Leave Balance: {alice_leave_balance}")

    # Get pending leave requests for Alice
    alice_pending_leaves = get_pending_leave_requests(employee_id="EMP001")
    print(f"\nAlice's Pending Leave Requests: {alice_pending_leaves}")

    # Get all pending leave requests
```

```
    all_pending_leaves = get_pending_leave_requests()
    print(f"\nAll Pending Leave Requests: {all_pending_leaves}")

    # Simulate submitting a new leave request
    print("\nSubmitting a new leave request for Charlie...")
    new_req = submit_leave_request("EMP003", "casual", "2025-09-01", "2025-09-
02", "Short break")
    print(f"New Request Submitted: {new_req}")
    print(f"Updated Leave Requests: {HR_DATABASE['leave_requests']}")
```

Input:

```
(Implicit inputs to functions)
- Employee ID: "EMP001", "EMP002", "EMP003"
- Employee Name: "Alice Smith", "Bob Johnson"
- New leave request details: employee_id="EMP003", type="casual",
start_date="2025-09-01", end_date="2025-09-02", reason="Short break"
```

Expected Output:

```
--- Accessing HR Data ---

Details for EMP001: {'id': 'EMP001', 'name': 'Alice Smith', 'department': 'HR',
'email': 'alice.s@example.com', 'leave_balance': {'sick': 10, 'casual': 15}}

Details for Bob Johnson: {'id': 'EMP002', 'name': 'Bob Johnson', 'department':
'Engineering', 'email': 'bob.j@example.com', 'leave_balance': {'sick': 8,
'casual': 12}}

Alice's Leave Balance: {'sick': 10, 'casual': 15}

Alice's Pending Leave Requests: [{'request_id': 'LR001', 'employee_id':
'EMP001', 'type': 'sick', 'start_date': '2025-06-01', 'end_date': '2025-06-03',
'status': 'pending', 'reason': 'Fever'}, {'request_id': 'LR003', 'employee_id':
'EMP001', 'type': 'casual', 'start_date': '2025-08-05', 'end_date': '2025-08-
05', 'status': 'pending', 'reason': 'Personal work'}]

All Pending Leave Requests: [{'request_id': 'LR001', 'employee_id': 'EMP001',
'type': 'sick', 'start_date': '2025-06-01', 'end_date': '2025-06-03', 'status':
'pending', 'reason': 'Fever'}, {'request_id': 'LR003', 'employee_id': 'EMP001',
'type': 'casual', 'start_date': '2025-08-05', 'end_date': '2025-08-05',
'status': 'pending', 'reason': 'Personal work'}]

Submitting a new leave request for Charlie...
New Request Submitted: {'request_id': 'LR004', 'employee_id': 'EMP003', 'type':
'casual', 'start_date': '2025-09-01', 'end_date': '2025-09-02', 'status':
'pending', 'reason': 'Short break'}
Updated Leave Requests: [{'request_id': 'LR001', 'employee_id': 'EMP001',
'type': 'sick', 'start_date': '2025-06-01', 'end_date': '2025-06-03', 'status':
'pending', 'reason': 'Fever'}, {'request_id': 'LR002', 'employee_id': 'EMP002',
'type': 'casual', 'start_date': '2025-07-10', 'end_date': '2025-07-12',
'status': 'approved', 'reason': 'Family event'}, {'request_id': 'LR003',
'employee_id': 'EMP001', 'type': 'casual', 'start_date': '2025-08-05',
'end_date': '2025-08-05', 'status': 'pending', 'reason': 'Personal work'},
{'request_id': 'LR004', 'employee_id': 'EMP003', 'type': 'casual', 'start_date':
'2025-09-01', 'end_date': '2025-09-02', 'status': 'pending', 'reason': 'Short
break'}]
```

Lab 8: Preprocess retrieved data to extract relevant information and prepare it for use in conversational interactions.

Title: Data Preprocessing for Conversational AI

Aim: To preprocess raw HR data retrieved from a data source (as simulated in Lab 7), extracting and formatting only the necessary information in a human-readable and conversation-friendly manner.

Procedure:

1. **Retrieve Raw Data:** Use the mock data access functions from Lab 7 to get raw employee or leave request data.
2. **Identify Key Information:** Determine which fields from the raw data are relevant for a conversational interaction (e.g., for leave balance: employee name, sick leave days, casual leave days; for leave request: type, dates, status).
3. **Format for Readability:** Convert raw data (e.g., dates, statuses) into a more natural language format.
4. **Structure for Output:** Organize the extracted information into a clear string or dictionary suitable for generating a bot response.
5. **Handle Missing Data:** Implement checks for missing data and provide appropriate messages.

Source Code: (Python - Data Preprocessing Functions)

```
# Re-using mock HR_DATABASE from Lab 7
HR_DATABASE = {
    "employees": [
        {"id": "EMP001", "name": "Alice Smith", "department": "HR", "email":
"alice.s@example.com", "leave_balance": {"sick": 10, "casual": 15}},
        {"id": "EMP002", "name": "Bob Johnson", "department": "Engineering",
"email": "bob.j@example.com", "leave_balance": {"sick": 8, "casual": 12}},
        {"id": "EMP003", "name": "Charlie Brown", "department": "Sales",
"email": "charlie.b@example.com", "leave_balance": {"sick": 5, "casual": 10}}
    ],
    "leave_requests": [
        {"request_id": "LR001", "employee_id": "EMP001", "type": "sick",
"start_date": "2025-06-01", "end_date": "2025-06-03", "status": "pending",
"reason": "Fever"},
        {"request_id": "LR002", "employee_id": "EMP002", "type": "casual",
"start_date": "2025-07-10", "end_date": "2025-07-12", "status": "approved",
"reason": "Family event"},
        {"request_id": "LR003", "employee_id": "EMP001", "type": "casual",
"start_date": "2025-08-05", "end_date": "2025-08-05", "status": "pending",
"reason": "Personal work"}
    ]
}

def get_employee_details(employee_id=None, employee_name=None):
    """
    Simulates fetching employee details from the HR database.
    """
    for emp in HR_DATABASE["employees"]:
        if employee_id and emp["id"].lower() == employee_id.lower():
            return emp
```

```
        if employee_name and emp["name"].lower() == employee_name.lower():
            return emp
    return None

def get_leave_balance(employee_id):
    """
    Simulates fetching an employee's leave balance.
    """
    employee = get_employee_details(employee_id=employee_id)
    if employee:
        return employee["leave_balance"]
    return None

def get_pending_leave_requests(employee_id=None):
    """
    Simulates fetching pending leave requests.
    If employee_id is provided, filters by employee.
    """
    pending_requests = []
    for req in HR_DATABASE["leave_requests"]:
        if req["status"].lower() == "pending":
            if employee_id is None or req["employee_id"].lower() ==
employee_id.lower():
                pending_requests.append(req)
    return pending_requests


def preprocess_leave_balance_data(employee_id):
    """
    Retrieves and preprocesses leave balance data for conversational output.
    """
    employee = get_employee_details(employee_id=employee_id)
    if not employee:
        return "I couldn't find an employee with that ID."

    balance = get_leave_balance(employee_id)
    if not balance:
        return f"Could not retrieve leave balance for {employee['name']}."

    response_parts = [f"{employee['name']}'s current leave balance:"]
    for leave_type, days in balance.items():
        response_parts.append(f"- {leave_type.capitalize()} Leave: {days} days")

    return "\n".join(response_parts)

def preprocess_pending_leave_requests(employee_id):
    """
    Retrieves and preprocesses pending leave requests for conversational output.
    """
    employee = get_employee_details(employee_id=employee_id)
    if not employee:
        return "I couldn't find an employee with that ID."

    pending_reqs = get_pending_leave_requests(employee_id=employee_id)
    if not pending_reqs:
        return f"There are no pending leave requests for {employee['name']}."

    response_parts = [f"Pending leave requests for {employee['name']}:"]
    for req in pending_reqs:
        response_parts.append(
            f"- Request ID: {req['request_id']}, Type:
{req['type'].capitalize()}, "
            f"From: {req['start_date']} to {req['end_date']}, Reason:
'{req['reason']}'"
        )
    return "\n".join(response_parts)
```

```
# Example usage:
if __name__ == "__main__":
    print("--- Preprocessing HR Data for Conversation ---")

    # Preprocess Alice's leave balance
    alice_balance_output = preprocess_leave_balance_data("EMP001")
    print(f"\nAlice's Leave Balance (Preprocessed):\n{alice_balance_output}")

    # Preprocess Bob's leave balance (missing employee)
    non_existent_employee_balance = preprocess_leave_balance_data("EMP999")
    print(f"\nNon-existent Employee Leave Balance
(Preprocessed):\n{non_existent_employee_balance}")

    # Preprocess Alice's pending leave requests
    alice_pending_output = preprocess_pending_leave_requests("EMP001")
    print(f"\nAlice's Pending Leave Requests
(Preprocessed):\n{alice_pending_output}")

    # Preprocess Bob's pending leave requests (none pending for Bob in mock
data)
    bob_pending_output = preprocess_pending_leave_requests("EMP002")
    print(f"\nBob's Pending Leave Requests
(Preprocessed):\n{bob_pending_output}")
```

Input:

```
(Implicit inputs to functions)
- Employee ID: "EMP001" (for existing employee)
- Employee ID: "EMP999" (for non-existent employee)
- Employee ID: "EMP002" (for employee with no pending requests)
```

Expected Output:

```
--- Preprocessing HR Data for Conversation ---

Alice's Leave Balance (Preprocessed):
Alice Smith's current leave balance:
- Sick Leave: 10 days
- Casual Leave: 15 days

Non-existent Employee Leave Balance (Preprocessed):
I couldn't find an employee with that ID.

Alice's Pending Leave Requests (Preprocessed):
Pending leave requests for Alice Smith:
- Request ID: LR001, Type: Sick, From: 2025-06-01 to 2025-06-03, Reason: 'Fever'
- Request ID: LR003, Type: Casual, From: 2025-08-05 to 2025-08-05, Reason:
'Personal work'

Bob's Pending Leave Requests (Preprocessed):
There are no pending leave requests for Bob Johnson.
```

Lab 9: Test the integration between the conversational AI system and HR systems to ensure data accuracy and consistency.

Title: Testing Conversational AI-HR System Integration

Aim: To design and execute test cases to verify the accurate and consistent exchange of data between the conversational AI system's logic and the simulated HR data sources.

Procedure:

1. **Define Test Scenarios:** Create specific test scenarios covering various interactions (e.g., requesting leave balance, submitting a new leave request, querying employee details).
2. **Expected Outcomes:** For each scenario, define the expected data retrieval or update in the HR system and the expected bot response.
3. **Simulate Interaction:** Write a script that simulates user inputs to the conversational AI and then directly checks the state of the mock HR database.
4. **Verification:** Compare the actual results from the HR system and the bot's response against the expected outcomes.
5. **Reporting:** Print clear pass/fail messages for each test case.

Source Code: (Python - Integration Test Script)

```
# Re-using mock HR_DATABASE and functions from Lab 7 & 8
HR_DATABASE = {
    "employees": [
        {"id": "EMP001", "name": "Alice Smith", "department": "HR", "email":
"alice.s@example.com", "leave_balance": {"sick": 10, "casual": 15}},
        {"id": "EMP002", "name": "Bob Johnson", "department": "Engineering",
"email": "bob.j@example.com", "leave_balance": {"sick": 8, "casual": 12}},
        {"id": "EMP003", "name": "Charlie Brown", "department": "Sales",
"email": "charlie.b@example.com", "leave_balance": {"sick": 5, "casual": 10}}
    ],
    "leave_requests": [
        {"request_id": "LR001", "employee_id": "EMP001", "type": "sick",
"start_date": "2025-06-01", "end_date": "2025-06-03", "status": "pending",
"reason": "Fever"},
        {"request_id": "LR002", "employee_id": "EMP002", "type": "casual",
"start_date": "2025-07-10", "end_date": "2025-07-12", "status": "approved",
"reason": "Family event"},
        {"request_id": "LR003", "employee_id": "EMP001", "type": "casual",
"start_date": "2025-08-05", "end_date": "2025-08-05", "status": "pending",
"reason": "Personal work"}
    ]
}

def get_employee_details(employee_id=None, employee_name=None):
    for emp in HR_DATABASE["employees"]:
        if employee_id and emp["id"].lower() == employee_id.lower():
            return emp
        if employee_name and emp["name"].lower() == employee_name.lower():
            return emp
    return None

def get_leave_balance(employee_id):
    employee = get_employee_details(employee_id=employee_id)
    if employee:
```

```
            return employee["leave_balance"]
    return None

def get_pending_leave_requests(employee_id=None):
    pending_requests = []
    for req in HR_DATABASE["leave_requests"]:
        if req["status"].lower() == "pending":
            if employee_id is None or req["employee_id"].lower() ==
employee_id.lower():
                pending_requests.append(req)
    return pending_requests

def submit_leave_request(employee_id, leave_type, start_date, end_date, reason):
    new_request_id = f"LR{len(HR_DATABASE['leave_requests']) + 1:03d}"
    new_request = {
        "request_id": new_request_id,
        "employee_id": employee_id,
        "type": leave_type,
        "start_date": start_date,
        "end_date": end_date,
        "status": "pending",
        "reason": reason
    }
    HR_DATABASE["leave_requests"].append(new_request)
    return new_request

def preprocess_leave_balance_data(employee_id):
    employee = get_employee_details(employee_id=employee_id)
    if not employee:
        return "I couldn't find an employee with that ID."
    balance = get_leave_balance(employee_id)
    if not balance:
        return f"Could not retrieve leave balance for {employee['name']}."
    response_parts = [f"{employee['name']}'s current leave balance:"]
    for leave_type, days in balance.items():
        response_parts.append(f"- {leave_type.capitalize()} Leave: {days} days")
    return "\n".join(response_parts)

def preprocess_pending_leave_requests(employee_id):
    employee = get_employee_details(employee_id=employee_id)
    if not employee:
        return "I couldn't find an employee with that ID."
    pending_reqs = get_pending_leave_requests(employee_id=employee_id)
    if not pending_reqs:
        return f"There are no pending leave requests for {employee['name']}."
    response_parts = [f"Pending leave requests for {employee['name']}:"]
    for req in pending_reqs:
        response_parts.append(
            f"- Request ID: {req['request_id']}, Type:
{req['type'].capitalize()}, "
            f"From: {req['start_date']} to {req['end_date']}, Reason:
'{req['reason']}'"
        )
    return "\n".join(response_parts)


def run_integration_tests():
    """Runs a series of integration tests."""
    print("--- Running Integration Tests ---")
    test_results = []

    # Test Case 1: Retrieve Alice's leave balance
    print("\nTest Case 1: Retrieve Alice's leave balance")
    expected_balance_str = "Alice Smith's current leave balance:\n- Sick Leave:
10 days\n- Casual Leave: 15 days"
    actual_balance_str = preprocess_leave_balance_data("EMP001")
    if actual_balance_str == expected_balance_str:
```

```
        test_results.append("Test Case 1: PASS - Alice's leave balance retrieved
correctly.")
    else:
        test_results.append(f"Test Case 1: FAIL -
Expected:\n{expected_balance_str}\nActual:\n{actual_balance_str}")

    # Test Case 2: Submit a new leave request for Charlie and verify in DB
    print("\nTest Case 2: Submit new leave request for Charlie and verify")
    initial_leave_requests_count = len(HR_DATABASE["leave_requests"])
    new_request_details = {
        "employee_id": "EMP003",
        "type": "annual",
        "start_date": "2025-10-01",
        "end_date": "2025-10-05",
        "reason": "Vacation"
    }
    submitted_request = submit_leave_request(**new_request_details)

    # Verify if the request was added to the mock DB
    if len(HR_DATABASE["leave_requests"]) == initial_leave_requests_count + 1
and \
       submitted_request in HR_DATABASE["leave_requests"]:
        test_results.append("Test Case 2: PASS - New leave request submitted and
found in DB.")
    else:
        test_results.append("Test Case 2: FAIL - New leave request submission
failed.")

    # Test Case 3: Retrieve Charlie's updated pending leave requests
    print("\nTest Case 3: Retrieve Charlie's updated pending leave requests")
    expected_charlie_pending_substr = "Type: Annual, From: 2025-10-01 to 2025-
10-05, Reason: 'Vacation'"
    actual_charlie_pending_str = preprocess_pending_leave_requests("EMP003")
    if expected_charlie_pending_substr in actual_charlie_pending_str:
        test_results.append("Test Case 3: PASS - Charlie's pending leave
requests updated correctly.")
    else:
        test_results.append(f"Test Case 3: FAIL - Expected substring
'{expected_charlie_pending_substr}' not found
in:\n{actual_charlie_pending_str}")

    # Test Case 4: Request for non-existent employee
    print("\nTest Case 4: Request for non-existent employee")
    expected_not_found_msg = "I couldn't find an employee with that ID."
    actual_not_found_msg = preprocess_leave_balance_data("EMP999")
    if actual_not_found_msg == expected_not_found_msg:
        test_results.append("Test Case 4: PASS - Handled non-existent employee
correctly.")
    else:
        test_results.append(f"Test Case 4: FAIL - Expected:
'{expected_not_found_msg}' Actual: '{actual_not_found_msg}'")

    print("\n--- Test Summary ---")
    for result in test_results:
        print(result)

# Run the tests
if __name__ == "__main__":
    run_integration_tests()
```

Input:

```
(Implicit inputs to the test functions)
- Test Case 1: Request for Alice's leave balance (EMP001)
- Test Case 2: Submission of a new annual leave request for Charlie (EMP003)
```

- Test Case 3: Request for Charlie's pending leave requests (EMP003) after submission
- Test Case 4: Request for a non-existent employee (EMP999)

Expected Output:

```
--- Running Integration Tests ---

Test Case 1: Retrieve Alice's leave balance
Test Case 2: Submit new leave request for Charlie and verify
Test Case 3: Retrieve Charlie's updated pending leave requests
Test Case 4: Request for non-existent employee

--- Test Summary ---
Test Case 1: PASS - Alice's leave balance retrieved correctly.
Test Case 2: PASS - New leave request submitted and found in DB.
Test Case 3: PASS - Charlie's pending leave requests updated correctly.
Test Case 4: PASS - Handled non-existent employee correctly.
```

Lab 10: Implement strategies to mitigate biases and ensure fairness and inclusivity in conversational AI systems.

Title: Mitigating Bias and Ensuring Fairness in Conversational AI

Aim: To understand and implement conceptual strategies to identify and mitigate biases in conversational AI systems, promoting fairness and inclusivity in interactions, particularly in HR contexts.

Procedure:

1. **Understand Sources of Bias:** Discuss how biases can enter AI systems (training data, algorithm design, human interaction patterns).
2. **Data Auditing:** Outline a process for auditing training data to identify and reduce demographic, linguistic, or historical biases.
3. **Fairness Metrics (Conceptual):** Briefly mention conceptual fairness metrics (e.g., demographic parity, equal opportunity) and how they could be applied.
4. **Response Diversification:** Implement strategies to diversify bot responses and avoid stereotypical language.
5. **Neutral Language:** Emphasize the use of neutral, respectful, and inclusive language in bot responses.
6. **User Feedback Loop:** Design a mechanism for collecting user feedback on perceived bias or unfairness.
7. **Example (Code based on response generation):** Provide a simple example of how to check for and correct potentially biased language in a generated response.

Source Code: (Python - Conceptual Bias Mitigation in Response Generation)

```
def generate_hr_response(intent, entities, data_from_hr_system=None):
    """
    Simulates generating an HR response, with a focus on bias mitigation.

    Args:
        intent (str): The recognized intent.
        entities (dict): Extracted entities.
        data_from_hr_system (any): Data retrieved from the HR system.

    Returns:
        str: The generated, bias-mitigated response.
    """
    response = ""

    if intent == "apply_leave":
        leave_type = entities.get("leave_type", "leave")
        start_date = entities.get("start_date", "a specified start date")
        end_date = entities.get("end_date", "a specified end date")
        response = f"Your request for {leave_type} from {start_date} to
{end_date} has been submitted. You will receive an update shortly."

    elif intent == "check_leave_balance":
        if data_from_hr_system:
            employee_name = data_from_hr_system.get("name", "your")
            sick_days = data_from_hr_system["leave_balance"].get("sick", 0)
            casual_days = data_from_hr_system["leave_balance"].get("casual", 0)
            response = f"{employee_name.capitalize()} current leave balance is:
{sick_days} sick days and {casual_days} casual days."
```

```
        else:
            response = "I'm sorry, I couldn't retrieve the leave balance at this
moment. Please try again later."

    elif intent == "onboarding_info":
        response = "Welcome! I can provide information about the onboarding
process. Please specify what details you need, such as required documents or
first-day instructions."

    else:
        response = "I'm here to assist with HR queries. How can I help you
today?"

    # --- Bias Mitigation Strategies (Conceptual Implementation) ---

    # 1. Neutral Language Check: Avoid gendered pronouns or assumptions
    # (This would typically involve more advanced NLP models or a rule-set)
    response = response.replace("he or she", "they")
    response = response.replace("his or her", "their")
    response = response.replace("manpower", "workforce")
    response = response.replace("chairman", "chairperson")

    # 2. Diversify phrasing for common responses to avoid monotony and perceived
bias
    if "submitted" in response and "update shortly" in response:
        alternative_phrases = [
            "Your request is now in process. An update will be sent soon.",
            "The system has recorded your request. You'll be notified of the
status.",
            "Your request has been logged. Expect a status update via email."
        ]
        # In a real system, you'd randomly pick one or use a more sophisticated
method
        # For this lab, we'll just show the concept.
        # response = random.choice(alternative_phrases) # Requires import random

    # 3. Add a disclaimer about AI limitations / fairness (optional but good
practice)
    if "I'm sorry" in response:
        response += " Please note that as an AI, I strive for fairness and
accuracy, but if you have concerns, please contact an HR representative."

    return response

# Mock HR Data (re-using from Lab 7)
MOCK_HR_DATA_ALICE = {"id": "EMP001", "name": "Alice Smith", "department": "HR",
"email": "alice.s@example.com", "leave_balance": {"sick": 10, "casual": 15}}

# Example usage:
if __name__ == "__main__":
    print("--- Generating Responses with Bias Mitigation Focus ---")

    # Scenario 1: Apply leave
    print("\nScenario: Apply Leave")
    response_1 = generate_hr_response("apply_leave", {"leave_type": "sick
leave", "start_date": "June 1st", "end_date": "June 3rd"})
    print(f"Bot: {response_1}")

    # Scenario 2: Check leave balance
    print("\nScenario: Check Leave Balance")
    response_2 = generate_hr_response("check_leave_balance", {},
MOCK_HR_DATA_ALICE)
    print(f"Bot: {response_2}")

    # Scenario 3: Onboarding info
    print("\nScenario: Onboarding Info")
    response_3 = generate_hr_response("onboarding_info", {})
```

```
    print(f"Bot: {response_3}")

    # Scenario 4: Unrecognized intent (triggering a 'sorry' message)
    print("\nScenario: Unrecognized Intent")
    response_4 = generate_hr_response("unrecognized", {})
    print(f"Bot: {response_4}")
```

Input:

```
(Implicit inputs to the `generate_hr_response` function)
- Intent: "apply_leave", Entities: {"leave_type": "sick leave", "start_date":
"June 1st", "end_date": "June 3rd"}
- Intent: "check_leave_balance", Entities: {}, Data: MOCK_HR_DATA_ALICE
- Intent: "onboarding_info", Entities: {}
- Intent: "unrecognized", Entities: {}
```

Expected Output:

```
--- Generating Responses with Bias Mitigation Focus ---

Scenario: Apply Leave
Bot: Your request for sick leave from June 1st to June 3rd has been submitted.
You will receive an update shortly.

Scenario: Check Leave Balance
Bot: Alice Smith current leave balance is: 10 sick days and 15 casual days.

Scenario: Onboarding Info
Bot: Welcome! I can provide information about the onboarding process. Please
specify what details you need, such as required documents or first-day
instructions.

Scenario: Unrecognized Intent
Bot: I'm here to assist with HR queries. How can I help you today? Please note
that as an AI, I strive for fairness and accuracy, but if you have concerns,
please contact an HR representative.
```

Lab 11: Build a Web hook for a Chatbot and connect with HR Systems.

Title: Building a Chatbot Webhook for HR System Integration

Aim: To understand the concept of webhooks and simulate building a simple webhook endpoint that a chatbot could use to communicate with and trigger actions in a backend HR system (represented by Python functions).

Procedure:

1. **Understand Webhooks:** Briefly explain what a webhook is and its role in real-time communication between systems.
2. **Simulate HR System Functions:** Re-use or adapt the HR data access/update functions from Lab 7.
3. **Create a Mock Webhook Endpoint:** Use a simple Python web framework (e.g., Flask) to create a basic HTTP endpoint (`/webhook`).
4. **Handle Incoming Requests:** Configure the endpoint to receive POST requests containing chatbot data (e.g., intent, entities).
5. **Process Data and Call HR Functions:** Inside the webhook, parse the incoming JSON payload, identify the intent, extract entities, and call the appropriate HR system function.
6. **Return Response:** Formulate a JSON response that the chatbot can understand (e.g., success/failure message, data to display).
7. **Testing:** Use a tool like `curl` or a simple Python script to simulate a POST request to the webhook.

Source Code: (Python - Flask Mock Webhook)

```
from flask import Flask, request, jsonify

app = Flask(__name__)

# Re-using mock HR_DATABASE and functions from previous labs
HR_DATABASE = {
    "employees": [
        {"id": "EMP001", "name": "Alice Smith", "department": "HR", "email":
"alice.s@example.com", "leave_balance": {"sick": 10, "casual": 15}},
        {"id": "EMP002", "name": "Bob Johnson", "department": "Engineering",
"email": "bob.j@example.com", "leave_balance": {"sick": 8, "casual": 12}},
        {"id": "EMP003", "name": "Charlie Brown", "department": "Sales",
"email": "charlie.b@example.com", "leave_balance": {"sick": 5, "casual": 10}}
    ],
    "leave_requests": [
        {"request_id": "LR001", "employee_id": "EMP001", "type": "sick",
"start_date": "2025-06-01", "end_date": "2025-06-03", "status": "pending",
"reason": "Fever"},
        {"request_id": "LR002", "employee_id": "EMP002", "type": "casual",
"start_date": "2025-07-10", "end_date": "2025-07-12", "status": "approved",
"reason": "Family event"},
        {"request_id": "LR003", "employee_id": "EMP001", "type": "casual",
"start_date": "2025-08-05", "end_date": "2025-08-05", "status": "pending",
"reason": "Personal work"}
    ]
}

def get_employee_details(employee_id=None, employee_name=None):
```

```
    for emp in HR_DATABASE["employees"]:
        if employee_id and emp["id"].lower() == employee_id.lower():
            return emp
        if employee_name and emp["name"].lower() == employee_name.lower():
            return emp
    return None

def get_leave_balance(employee_id):
    employee = get_employee_details(employee_id=employee_id)
    if employee:
        return employee["leave_balance"]
    return None

def submit_leave_request(employee_id, leave_type, start_date, end_date, reason):
    new_request_id = f"LR{len(HR_DATABASE['leave_requests']) + 1:03d}"
    new_request = {
        "request_id": new_request_id,
        "employee_id": employee_id,
        "type": leave_type,
        "start_date": start_date,
        "end_date": end_date,
        "status": "pending",
        "reason": reason
    }
    HR_DATABASE["leave_requests"].append(new_request)
    return new_request

def preprocess_leave_balance_data(employee_id):
    employee = get_employee_details(employee_id=employee_id)
    if not employee:
        return "I couldn't find an employee with that ID."
    balance = get_leave_balance(employee_id)
    if not balance:
        return f"Could not retrieve leave balance for {employee['name']}."
    response_parts = [f"{employee['name']}'s current leave balance:"]
    for leave_type, days in balance.items():
        response_parts.append(f"- {leave_type.capitalize()} Leave: {days} days")
    return "\n".join(response_parts)

@app.route('/webhook', methods=['POST'])
def webhook():
    """
    Webhook endpoint to receive chatbot requests and interact with HR systems.
    """
    req = request.get_json(silent=True, force=True)
    print(f"Received Webhook Request: {req}")

    intent = req.get('queryResult', {}).get('intent', {}).get('displayName')
    parameters = req.get('queryResult', {}).get('parameters', {})
    employee_id = parameters.get('employee_id') # Assume employee_id is passed
as a parameter

    fulfillment_text = "I'm sorry, I couldn't fulfill your request."

    if intent == 'CheckLeaveBalance':
        if employee_id:
            fulfillment_text = preprocess_leave_balance_data(employee_id)
        else:
            fulfillment_text = "Please provide your employee ID to check your
leave balance."
    elif intent == 'SubmitLeaveRequest':
        emp_id = parameters.get('employee_id')
        leave_type = parameters.get('leave_type')
        start_date = parameters.get('start_date')
        end_date = parameters.get('end_date')
        reason = parameters.get('reason', 'No reason provided')
```

```
        if emp_id and leave_type and start_date and end_date:
            try:
                submitted_req = submit_leave_request(emp_id, leave_type,
start_date, end_date, reason)
                fulfillment_text = f"Your {leave_type} request (ID:
{submitted_req['request_id']}) from {start_date} to {end_date} has been
submitted for approval."
            except Exception as e:
                fulfillment_text = f"An error occurred while submitting your
request: {str(e)}"
        else:
            fulfillment_text = "I need more details to submit your leave request
(employee ID, leave type, start date, end date)."
    elif intent == 'Welcome':
        fulfillment_text = "Hello! How can I assist you with HR today?"
    else:
        fulfillment_text = "I'm not sure how to handle that request. Please try
asking about leave or employee information."

    return jsonify({
        'fulfillmentText': fulfillment_text
    })

if __name__ == '__main__':
    # To run this, save it as app.py and run `flask run` in your terminal.
    # You would then expose it via a tool like ngrok for external chatbot
integration.
    print("Flask app is running. Access the webhook at
http://127.0.0.1:5000/webhook")
    app.run(debug=True) # debug=True allows for automatic reloading on code
changes
```

Input: (Simulated POST requests to the `/webhook` endpoint)

```
# Example 1: Check Leave Balance
{
  "queryResult": {
    "intent": {
      "displayName": "CheckLeaveBalance"
    },
    "parameters": {
      "employee_id": "EMP001"
    }
  }
}

# Example 2: Submit Leave Request
{
  "queryResult": {
    "intent": {
      "displayName": "SubmitLeaveRequest"
    },
    "parameters": {
      "employee_id": "EMP002",
      "leave_type": "casual",
      "start_date": "2025-11-10",
      "end_date": "2025-11-12",
      "reason": "Personal errands"
    }
  }
}

# Example 3: Welcome Intent
{
  "queryResult": {
```

```
    "intent": {
      "displayName": "Welcome"
    },
    "parameters": {}
  }
}

# Example 4: Unknown Intent
{
  "queryResult": {
    "intent": {
      "displayName": "UnknownIntent"
    },
    "parameters": {}
  }
}
```

Expected Output: (JSON responses from the webhook)

```
# For Example 1 (Check Leave Balance):
{
  "fulfillmentText": "Alice Smith's current leave balance:\n- Sick Leave: 10
days\n- Casual Leave: 15 days"
}

# For Example 2 (Submit Leave Request):
{
  "fulfillmentText": "Your casual request (ID: LR004) from 2025-11-10 to 2025-
11-12 has been submitted for approval."
}

# For Example 3 (Welcome Intent):
{
  "fulfillmentText": "Hello! How can I assist you with HR today?"
}

# For Example 4 (Unknown Intent):
{
  "fulfillmentText": "I'm not sure how to handle that request. Please try asking
about leave or employee information."
}
```

(Note: To run this code, you need to have Flask installed (`pip install Flask`) and run the script as a Flask application. You would then use `curl` or a similar tool to send POST requests to `http://127.0.0.1:5000/webhook`.)

Lab 12: Conduct user testing sessions with human participants to evaluate the usability and effectiveness of the conversational AI system.

Title: User Testing and Evaluation of Conversational AI

Aim: To outline a methodology for conducting user testing sessions to evaluate the usability, effectiveness, and user satisfaction of a conversational AI system in an HR context.

Procedure:

1. **Define Objectives:** Clearly state what aspects of the chatbot are being evaluated (e.g., intent recognition accuracy, response clarity, task completion rate, user satisfaction).
2. **Recruit Participants:** Select a diverse group of target users (e.g., employees, HR staff).
3. **Develop Test Scenarios/Tasks:** Create realistic HR-related tasks for participants to complete using the chatbot (e.g., "Find out your sick leave balance," "Submit a casual leave request for next week," "Ask about onboarding documents").
4. **Choose Metrics:**
    - **Quantitative:** Task completion rate, time on task, number of turns per task, error rate.
    - **Qualitative:** User satisfaction (Likert scale), perceived helpfulness, ease of use, feedback on specific issues.
5. **Conduct Sessions:**
    - **Introduction:** Explain the purpose and process.
    - **Task Execution:** Observe participants as they interact with the chatbot, encouraging them to think aloud.
    - **Post-Test Survey/Interview:** Gather qualitative feedback.
6. **Analyze Results:** Compile and analyze both quantitative and qualitative data.
7. **Identify Improvements:** Based on findings, list actionable recommendations for improving the chatbot.

Source Code: (Conceptual - User Testing Plan/Report Outline)

```
### User Testing Plan Outline

**1. Testing Objectives:**
* Evaluate the accuracy of intent recognition for core HR queries (e.g., leave, onboarding).
* Assess the clarity and helpfulness of the chatbot's responses.
* Measure the task completion rate for common HR self-service tasks.
* Gather qualitative feedback on overall user experience and satisfaction.

**2. Participant Recruitment:**
* Target Audience: Employees (new hires, existing staff), HR administrators.
* Number of Participants: 5-10 (for initial qualitative insights).
* Recruitment Method: Internal email, sign-up sheet.

**3. Test Scenarios/Tasks:**
* **Task 1: Leave Balance Inquiry**
    * *Prompt:* "Find out how many casual leaves you have remaining."
* **Task 2: Sick Leave Submission**
    * *Prompt:* "Submit a sick leave request for 2 days next month, providing a reason."
* **Task 3: Onboarding Document Information**
```

```
    * *Prompt:* "Ask the chatbot what documents a new employee needs to submit
during onboarding."
* **Task 4: General HR Policy Question**
    * *Prompt:* "Inquire about the company's policy on remote work." (Expected:
Out of scope or general guidance)

**4. Data Collection Methods:**
* **Observation:** Note down user utterances, bot responses, navigation paths,
points of confusion, and errors.
* **System Logs:** Record conversation transcripts, detected intents, and
extracted entities.
* **Post-Task Survey:**
    * "On a scale of 1-5, how easy was it to complete this task?"
    * "On a scale of 1-5, how helpful was the chatbot's response?"
    * "Did you encounter any frustrating moments? If so, please describe."
    * "What did you like most/least about interacting with the chatbot?"
    * "What improvements would you suggest?"

**5. Metrics to Track:**
* **Quantitative:**
    * Task Completion Rate (per task)
    * Average Turns per Task
    * Error Rate (e.g., number of times fallback intent triggered)
    * Survey scores (average ease, helpfulness)
* **Qualitative:**
    * Common pain points/frustrations
    * Unexpected user behaviors
    * Suggestions for new features or content

### Sample User Feedback Form (Textual)

**User Testing Feedback Form**

**Participant ID:** _________
**Date:** _________

**Overall Experience:**
1.  How easy was it to use the HR chatbot? (1=Very Difficult, 5=Very Easy)
    [ ] 1  [ ] 2  [ ] 3  [ ] 4  [ ] 5
2.  How helpful were the chatbot's responses? (1=Not Helpful, 5=Very Helpful)
    [ ] 1  [ ] 2  [ ] 3  [ ] 4  [ ] 5
3.  Did you feel the chatbot understood your requests accurately?
(Yes/No/Sometimes)
    [ ] Yes  [ ] No  [ ] Sometimes

**Specific Task Feedback:**
* **Task: [Task Name, e.g., "Leave Balance Inquiry"]**
    * Did you successfully complete this task? [ ] Yes  [ ] No
    * If no, why not?
    * What could have made this task easier?

**Open-Ended Feedback:**
* What did you like most about the HR chatbot?
* What did you like least about the HR chatbot?
* What improvements would you suggest?
* Are there any HR tasks you wish the chatbot could handle that it currently
doesn't?

Thank you for your valuable feedback!
```

Input: N/A (Conceptual Design - User interactions during live testing)

Expected Output: A detailed outline of a user testing plan, including objectives, participant recruitment, test scenarios, data collection methods, and metrics, along with a sample feedback form.

Lab 13: Project integrating concepts learned throughout the course.

Title: Integrated Conversational AI HR Assistant Project

Aim: To design and implement a comprehensive conversational AI HR assistant that integrates all previously learned concepts, including tokenization, POS tagging, intent recognition, dialogue management, error handling, and interaction with a simulated HR backend.

Procedure:

1. **Define Scope:** Choose 2-3 core HR functionalities (e.g., Leave Management, Employee Information Lookup, Basic Onboarding FAQs).
2. **System Architecture:** Design a high-level architecture diagram (textual) showing the components: User Interface (simulated), NLP module (Intent/Entity), Dialogue Manager, HR Backend Integrator, HR Data Source.
3. **Component Implementation:**
    - **NLP Module:** Re-use/refine tokenization, POS tagging, and intent recognition (Lab 1, 2, 6).
    - **Dialogue Manager:** Implement state management, conversational flows (Lab 3, 4), and error handling (Lab 5).
    - **HR Backend Integrator:** Use mock HR data and functions (Lab 7, 8) to simulate data access and updates.
    - **Response Generation:** Combine retrieved data with natural language generation.
4. **Integration:** Connect all modules to work cohesively.
5. **Testing:** Perform end-to-end testing with various user inputs, including edge cases.

Source Code: (Python - High-Level Integration of Modules)

```
# --- Core Modules (Simplified Re-use from previous labs) ---
import re

# Lab 1: Tokenization
def tokenize_text(text):
    token_pattern = re.compile(r'\w+|[^\w\s]')
    return token_pattern.findall(text)

# Lab 6: Simple Intent Recognition
def recognize_intent(text):
    text_lower = text.lower()
    if "apply" in text_lower and ("leave" in text_lower or "vacation" in
text_lower):
        return "apply_leave"
    elif "check" in text_lower and ("balance" in text_lower or "leaves" in
text_lower):
        return "check_leave_balance"
    elif "onboarding" in text_lower or "new employee" in text_lower or "join" in
text_lower:
        return "onboarding_info"
    elif "hi" in text_lower or "hello" in text_lower:
        return "greet"
    elif "bye" in text_lower or "goodbye" in text_lower:
        return "farewell"
    else:
        return "unrecognized"
```

```
# Simple Entity Extraction (for demo purposes)
def extract_entities(text):
    entities = {}
    text_lower = text.lower()
    if "sick leave" in text_lower: entities["leave_type"] = "sick"
    if "casual leave" in text_lower: entities["leave_type"] = "casual"
    if "annual leave" in text_lower: entities["leave_type"] = "annual"
    # Placeholder for date extraction
    if "tomorrow" in text_lower: entities["start_date"] = "2025-05-23" # Example
date
    if "next week" in text_lower: entities["start_date"] = "2025-05-26" #
Example date
    # Simple employee ID/name extraction
    match_id = re.search(r'EMP(\d+)', text_lower)
    if match_id: entities["employee_id"] = match_id.group(0).upper()
    return entities

# Lab 7 & 8: Mock HR Backend & Preprocessing
HR_DATABASE = {
    "employees": [
        {"id": "EMP001", "name": "Alice Smith", "department": "HR", "email":
"alice.s@example.com", "leave_balance": {"sick": 10, "casual": 15}},
        {"id": "EMP002", "name": "Bob Johnson", "department": "Engineering",
"email": "bob.j@example.com", "leave_balance": {"sick": 8, "casual": 12}},
    ],
    "leave_requests": [
        {"request_id": "LR001", "employee_id": "EMP001", "type": "sick",
"start_date": "2025-06-01", "end_date": "2025-06-03", "status": "pending",
"reason": "Fever"},
    ]
}

def get_employee_details(employee_id=None, employee_name=None):
    for emp in HR_DATABASE["employees"]:
        if employee_id and emp["id"].lower() == employee_id.lower(): return emp
        if employee_name and emp["name"].lower() == employee_name.lower():
return emp
    return None

def get_leave_balance(employee_id):
    employee = get_employee_details(employee_id=employee_id)
    return employee["leave_balance"] if employee else None

def submit_leave_request(employee_id, leave_type, start_date, end_date, reason):
    new_request_id = f"LR{len(HR_DATABASE['leave_requests']) + 1:03d}"
    new_request = {"request_id": new_request_id, "employee_id": employee_id,
"type": leave_type, "start_date": start_date, "end_date": end_date, "status":
"pending", "reason": reason}
    HR_DATABASE["leave_requests"].append(new_request)
    return new_request

def preprocess_leave_balance_data(employee_id):
    employee = get_employee_details(employee_id=employee_id)
    if not employee: return "I couldn't find an employee with that ID."
    balance = get_leave_balance(employee_id)
    response_parts = [f"{employee['name']}'s current leave balance:"]
    for leave_type, days in balance.items(): response_parts.append(f"-
{leave_type.capitalize()} Leave: {days} days")
    return "\n".join(response_parts)

# --- Dialogue Manager (Integrating NLP and Backend) ---
class HRBot:
    def __init__(self):
        self.context = {} # Stores conversation state for multi-turn
interactions

    def process_message(self, user_input):
```

```
        tokens = tokenize_text(user_input)
        intent = recognize_intent(user_input)
        entities = extract_entities(user_input)

        response = ""

        if intent == "greet":
            response = "Hello! How can I assist you with HR today?"
        elif intent == "farewell":
            response = "Goodbye! Have a great day."
            self.context = {} # Clear context on farewell
        elif intent == "apply_leave":
            return self._handle_apply_leave(entities)
        elif intent == "check_leave_balance":
            return self._handle_check_leave_balance(entities)
        elif intent == "onboarding_info":
            response = "I can provide information about the onboarding process.
What specific details are you looking for?"
        else: # Unrecognized intent / Error handling
            response = "I'm sorry, I didn't understand that. I can help with
leave requests, checking balances, or onboarding information. Can you rephrase?"

        return response

    def _handle_apply_leave(self, entities):
        employee_id = entities.get("employee_id") or
self.context.get("employee_id")
        leave_type = entities.get("leave_type") or
self.context.get("leave_type")
        start_date = entities.get("start_date") or
self.context.get("start_date")
        end_date = entities.get("end_date") or self.context.get("end_date")
        reason = entities.get("reason") # Assume reason is extracted if provided

        # Update context
        self.context.update(entities)

        if not employee_id:
            return "To apply for leave, I need your employee ID. Can you please
provide it?"
        if not leave_type:
            return "What type of leave are you applying for (sick, casual,
annual)?"
        if not start_date:
            return "What is the start date for your leave?"
        if not end_date:
            return "What is the end date for your leave?"

        # All required info collected, proceed to submit
        try:
            submitted_req = submit_leave_request(employee_id, leave_type,
start_date, end_date, reason)
            self.context = {} # Clear context after successful submission
            return f"Your {leave_type} leave request (ID:
{submitted_req['request_id']}) from {start_date} to {end_date} has been
submitted for approval."
        except Exception as e:
            return f"An error occurred while submitting your request: {str(e)}.
Please try again."

    def _handle_check_leave_balance(self, entities):
        employee_id = entities.get("employee_id") or
self.context.get("employee_id")

        if not employee_id:
            return "To check your leave balance, I need your employee ID. Can
you please provide it?"
```

```
        # Retrieve and preprocess data
        balance_info = preprocess_leave_balance_data(employee_id)
        self.context = {} # Clear context after successful query
        return balance_info

# --- Main Interaction Loop ---
if __name__ == "__main__":
    bot = HRBot()
    print("HR Chatbot: Hello! How can I assist you with HR today? (Type 'exit'
to quit)")

    while True:
        user_input = input("You: ")
        if user_input.lower() == 'exit':
            print("HR Chatbot: Goodbye!")
            break

        response = bot.process_message(user_input)
        print(f"HR Chatbot: {response}")
```

Input:

```
(Interactive conversation with the bot)
You: Hi
You: I want to apply for leave.
You: My ID is EMP001
You: Sick leave
You: From tomorrow to May 25th
You: What is my leave balance?
You: My ID is EMP002
You: Tell me about onboarding.
You: exit
```

Expected Output:

```
HR Chatbot: Hello! How can I assist you with HR today? (Type 'exit' to quit)
You: Hi
HR Chatbot: Hello! How can I assist you with HR today?
You: I want to apply for leave.
HR Chatbot: To apply for leave, I need your employee ID. Can you please provide
it?
You: My ID is EMP001
HR Chatbot: What type of leave are you applying for (sick, casual, annual)?
You: Sick leave
HR Chatbot: What is the start date for your leave?
You: From tomorrow to May 25th
HR Chatbot: Your sick leave request (ID: LR002) from 2025-05-23 to May 25th has
been submitted for approval.
You: What is my leave balance?
HR Chatbot: To check your leave balance, I need your employee ID. Can you please
provide it?
You: My ID is EMP002
HR Chatbot: Bob Johnson's current leave balance:
- Sick Leave: 8 days
- Casual Leave: 12 days
You: Tell me about onboarding.
HR Chatbot: I can provide information about the onboarding process. What
specific details are you looking for?
You: exit
HR Chatbot: Goodbye!
```

Lab 14: Demonstrating the functionality and effectiveness of the conversational AI system.

Title: Demonstrating Conversational AI System Functionality

Aim: To prepare and execute a demonstration of the integrated conversational AI HR assistant, showcasing its key functionalities, effectiveness in handling common HR queries, and user experience.

Procedure:

1. **Prepare Demo Script:** Create a script outlining specific user interactions and the expected bot responses to highlight the system's capabilities (e.g., successful leave application, leave balance check, handling of invalid input).
2. **Highlight Key Features:** Emphasize intent recognition, entity extraction, dialogue management, and integration with the HR backend.
3. **Showcase Error Handling:** Include scenarios where the bot gracefully handles misunderstandings or missing information.
4. **User Interface (Conceptual):** Describe how the interaction would look in a simple chat interface.
5. **Performance Metrics (Discuss):** Briefly discuss how the system's effectiveness (e.g., accuracy, speed) could be measured in a real-world scenario.
6. **Q&A Preparation:** Anticipate potential questions and prepare answers regarding the system's design, limitations, and future enhancements.

Source Code: (Conceptual - Demonstration Script/Narrative)

```
### Demonstration Script for HR Conversational AI

**Goal:** To showcase the HR Chatbot's ability to handle common employee
requests and provide HR information efficiently.

**Participants:** Presenter, Audience (simulated users)

---

**1. Introduction (Presenter)**
* "Good morning/afternoon, everyone. Today, we're excited to demonstrate our
Conversational AI HR Assistant, designed to streamline common HR queries and
tasks for employees."
* "This system integrates natural language understanding, dialogue management,
and a simulated HR backend to provide a seamless experience."

**2. Core Functionality Demo**

    **Scenario A: Checking Leave Balance**
    * **Presenter:** "Let's start with a common request: checking an employee's
leave balance."
    * **Simulated User Input (on screen/typed):** "Hi, I'm EMP001. How many
leaves do I have?"
    * **Expected Bot Response:** "Alice Smith's current leave balance: - Sick
Leave: 10 days - Casual Leave: 15 days"
    * **Presenter:** "As you can see, the bot correctly identified the employee
and retrieved their real-time leave balance from our HR system."

    **Scenario B: Submitting a Leave Request (Multi-turn)**
```

```
    * **Presenter:** "Now, let's try submitting a leave request, which often
involves multiple pieces of information."
    * **Simulated User Input:** "I want to apply for leave."
    * **Expected Bot Response:** "To apply for leave, I need your employee ID.
Can you please provide it?"
    * **Simulated User Input:** "My ID is EMP002."
    * **Expected Bot Response:** "What type of leave are you applying for (sick,
casual, annual)?"
    * **Simulated User Input:** "Casual leave."
    * **Expected Bot Response:** "What is the start date for your leave?"
    * **Simulated User Input:** "From next Monday to next Wednesday."
    * **Expected Bot Response:** "Your casual leave request (ID: LRxxx) from
[Start Date] to [End Date] has been submitted for approval."
    * **Presenter:** "The bot intelligently guided the user through the
necessary steps, collecting all required information before submitting the
request to the HR system."

**3. Error Handling Demonstration**

    **Scenario C: Unrecognized Intent**
    * **Presenter:** "What happens if a user asks something outside the bot's
scope or in an unclear way?"
    * **Simulated User Input:** "Tell me a joke."
    * **Expected Bot Response:** "I'm sorry, I didn't understand that. I can
help with leave requests, checking balances, or onboarding information. Can you
rephrase?"
    * **Presenter:** "The bot politely informs the user of its limitations and
redirects them to relevant topics, preventing frustration."

    **Scenario D: Missing Information**
    * **Presenter:** "If a user provides incomplete information, the bot will
prompt them for what's missing."
    * **Simulated User Input:** "Apply for leave."
    * **Expected Bot Response:** "To apply for leave, I need your employee ID.
Can you please provide it?"
    * **Presenter:** "This ensures all necessary data is collected before
processing the request."

**4. Conclusion (Presenter)**
* "This demonstration highlights how our Conversational AI HR Assistant can
significantly improve employee self-service, reduce HR workload, and provide
instant support."
* "We've focused on accuracy, usability, and robust error handling to ensure a
positive user experience."
* "We are continuously working on expanding its capabilities and refining its
understanding."

**5. Q&A Session**
* "Are there any questions about the system's functionality, its integration, or
future plans?"
```

Input: N/A (Conceptual Design - Live demonstration with simulated user inputs)

Expected Output: A detailed demonstration script or narrative that outlines the flow of a presentation, showcasing the chatbot's functionalities, error handling, and overall effectiveness.

Lab 15: Deploying a chatbot on Heroku.

Title: Deploying a Chatbot Webhook on Heroku

Aim: To understand the process of deploying a Python-based chatbot webhook (like the one developed in Lab 11) to a cloud platform like Heroku, making it accessible over the internet.

Procedure:

1. **Heroku Account Setup:** Create a Heroku account and install the Heroku CLI.
2. **Prepare Python Application:**
    - Ensure your Flask webhook application (from Lab 11) is in a file (e.g., `app.py`).
    - Create a `requirements.txt` file listing all Python dependencies (e.g., `Flask`).
    - Create a `Procfile` to tell Heroku how to run your application (e.g., `web: gunicorn app:app`).
    - (Optional but recommended) Use `gunicorn` for a production-ready WSGI server (`pip install gunicorn`).
3. **Git Initialization:** Initialize a Git repository in your project directory.
4. **Heroku App Creation:** Create a new Heroku application using the Heroku CLI.
5. **Deployment:** Push your local Git repository to the Heroku remote.
6. **Verification:** Check the Heroku logs and access the deployed webhook URL to confirm it's running.
7. **Connect to Chatbot Platform:** (Conceptual) Explain how to configure a chatbot platform (e.g., Dialogflow, Rasa) to use this deployed webhook URL.

Source Code: (Conceptual - Files required for Heroku Deployment)

```
# --- File: app.py (Example Flask Webhook from Lab 11) ---
# from flask import Flask, request, jsonify
# import re
# # ... (Include all HR_DATABASE and functions from Lab 11) ...
# @app.route('/webhook', methods=['POST'])
# def webhook():
#     # ... (Webhook logic from Lab 11) ...
#     return jsonify({'fulfillmentText': fulfillment_text})
# if __name__ == '__main__':
#     app.run(debug=True)
# ------------------------------------------------------------

# --- File: requirements.txt ---
# This file lists Python dependencies for Heroku to install.
Flask==2.3.2
gunicorn==21.2.0
# Add any other libraries your app uses, e.g., nltk, scikit-learn etc.
# If using NLTK, you might need a custom buildpack or ensure data is downloaded
on deploy.

# --- File: Procfile ---
# This file tells Heroku how to run your web application.
# `web`: indicates a web process
# `gunicorn`: the WSGI HTTP server
# `app:app`: tells gunicorn to look for the 'app' object in the 'app.py' file
web: gunicorn app:app

# --- File: .gitignore ---
# This file tells Git which files/directories to ignore.
__pycache__/
```

```
*.pyc
.env
venv/
*.sqlite3
# Add any other files you don't want to commit to Git
```

Input: (CLI Commands for Deployment)

```
# 1. Create a virtual environment (optional but recommended)
python -m venv venv
source venv/bin/activate # On Windows: .\venv\Scripts\activate

# 2. Install dependencies
pip install Flask gunicorn

# 3. Create requirements.txt (if not already created)
pip freeze > requirements.txt

# 4. Initialize Git repository
git init
git add .
git commit -m "Initial commit for Heroku deployment"

# 5. Create Heroku app
heroku create your-unique-chatbot-name # Replace 'your-unique-chatbot-name' with
a unique name

# 6. Deploy to Heroku
git push heroku main

# 7. Open the deployed app in your browser (optional, for basic check)
heroku open

# 8. Check logs for any issues
heroku logs --tail
```

Expected Output:

```
# Output from `heroku create`:
Creating ⬢ your-unique-chatbot-name... done
https://your-unique-chatbot-name.herokuapp.com/ | https://git.heroku.com/your-
unique-chatbot-name.git

# Output from `git push heroku main`:
Counting objects: 12, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (12/12), 1.25 KiB | 1.25 MiB/s, done.
Total 12 (delta 0), reused 0 (delta 0), pack-reused 0
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Building on the Heroku stack...
remote: -----> Python app detected
remote: -----> Installing python-3.x.x
remote: -----> Installing pip
remote: -----> Installing dependencies with pip
remote:        ... (dependency installation logs) ...
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done, slug size is X.XMB
```

```
remote: -----> Launching...
remote:        Released vX
remote:        https://your-unique-chatbot-name.herokuapp.com/ deployed to
Heroku
remote:
To https://git.heroku.com/your-unique-chatbot-name.git
 * [new branch]      main -> main

# Output from `heroku logs --tail` (showing app starting up):
2025-05-22T10:00:00.000000+00:00 app[web.1]: [2025-05-22 10:00:00 +0000] [1]
[INFO] Starting gunicorn 21.2.0
2025-05-22T10:00:00.000000+00:00 app[web.1]: [2025-05-22 10:00:00 +0000] [1]
[INFO] Listening at: http://0.0.0.0:XXXX (1)
2025-05-22T10:00:00.000000+00:00 app[web.1]: [2025-05-22 10:00:00 +0000] [1]
[INFO] Using worker: sync
2025-05-22T10:00:00.000000+00:00 app[web.1]: [2025-05-22 10:00:00 +0000] [5]
[INFO] Booting worker with pid: 5
```

(Note: The exact output will vary based on your Heroku app name, Python version, and deployment details.)