The University of Technology, Jamaica

The School of Computing and Information Technology

CIT4004 – Analysis of Programming Languages

Lecturer: Dr. David White

Kemar Christie 2005904

Roberto Davis 2100101

Dwayne Gibbs 2007512

Tyoni Davis 1701860

Danielle Jones 1900398

April 5, 2025

Group Project

Project Report

- **Paradigm the language you developed belongs to.**

  The programming language developed in this project adheres to the procedural programming paradigm. This paradigm is centered around the concept of procedure calls, where the program is structured into procedures or routines that operate on data. It emphasizes a step-by-step sequence of instructions, making it suitable for tasks that require a clear and logical flow of control.

- **Explaining whether your language is general purpose or domain specific.**

  Our booking programming language is domain-specific. It is designed solely to facilitate booking operations such as scheduling, availability tracking, reservation processing, and user management. Because it focuses on this specialized area and lacks the broad capabilities of a general-purpose language, it cannot be used effectively outside the booking domain.

- **Explaining whether your language is low level or high level.**

  The programming language developed in this project is classified as a high-level language. This classification is based on the fact that it was implemented using Python, a widely recognized high-level programming language, along with parsing tools such as PLY (Python Lex-Yacc). As such, our language prioritizes readability, maintainability, and ease of use , hallmarks of high-level language design.

- **Correct grammar for the language you developed.**

  // Main command types

```
START    →    COMMAND SYMBOL

COMMAND    →  BOOKING_COMMAND
              | LIST_COMMAND
              | PAYMENT_COMMAND
              | INQUIRY_COMMAND
              | RENT_COMMAND
              | CONFIRMATION_COMMAND
              | CANCELLATION_COMMAND

LIST_COMMAND    →  LIST_KEYWORD RESOURCE LOCATION_MARKER DEPARTURE LOCATION_MARKER ARRIVAL SYMBOL
                   | LIST_KEYWORD CONTEXT_KEYWORD RENT_KEYWORD RESOURCE LOCATION_MARKER LOCATION SYMBOL
                   | LIST_KEYWORD SERVICE CONTEXT_KEYWORD SYMBOL
                   | LIST_KEYWORD SERVICE CONTEXT_KEYWORD LOCATION_MARKER DEPARTURE LOCATION_MARKER ARRIVAL SYMBOL
                   | LIST_KEYWORD CONTEXT_KEYWORD USERNAME SYMBOL

BOOKING_COMMAND    →    ACTION_KEYWORD RESOURCE LOCATION_MARKER DEPARTURE LOCATION_MARKER ARRIVAL SYMBOL
          | ACTION_KEYWORD RESOURCE LOCATION_MARKER ARRIVAL LOCATION_MARKER DEPARTURE SYMBOL
          | ACTION_KEYWORD RESOURCE LOCATION_MARKER ARRIVAL LOCATION_MARKER DEPARTURE CONNECTIVE_WORD CONTEXT_KEYWORD CONDITIONS MONEY SYMBOL
          | ACTION_KEYWORD RESOURCE LOCATION_MARKER ARRIVAL LOCATION_MARKER DEPARTURE ARTICLE_CONJUNCTION ARTICLE_CONJUNCTION RESOURCE LOCATION_MARKER START_DATE LOCATION_MARKER END_DATE SYMBOL
          | ACTION_KEYWORD SERVICE RESOURCE LOCATION_MARKER DEPARTURE LOCATION_MARKER ARRIVAL CONTEXT_KEYWORD START_DATE LOCATION_MARKER TIME CONTEXT_KEYWORD USERNAME SYMBOL
          | ACTION_KEYWORD RESOURCE LOCATION_MARKER DEPARTURE LOCATION_MARKER ARRIVAL CONTEXT_KEYWORD START_DATE LOCATION_MARKER TIME CONTEXT_KEYWORD CONTEXT_KEYWORD END_DATE LOCATION_MARKER
TIME SYMBOL
          | ACTION_KEYWORD RESOURCE LOCATION_MARKER SERVICE LOCATION_MARKER START_DATE LOCATION_MARKER END_DATE CONTEXT_KEYWORD USERNAME SYMBOL
          | ACTION_KEYWORD RESOURCE LOCATION_MARKER DEPARTURE CONTEXT_KEYWORD DATE ARRIVAL CONTEXT_KEYWORD DATE SYMBOL
          | ACTION_KEYWORD RESOURCE LOCATION_MARKER LOCATION LOCATION_MARKER START_DATE LOCATION_MARKER END_DATE CONTEXT_KEYWORD USERNAME SYMBOL
          | ACTION_KEYWORD SERVICE RESOURCE LOCATION_MARKER DEPARTURE LOCATION_MARKER ARRIVAL CONTEXT_KEYWORD START_DATE LOCATION_MARKER TIME CONTEXT_KEYWORD NUMBER PASSENGER_TYPE SYMBOL
          | ACTION_KEYWORD NUMBER RESOURCE CONTEXT_KEYWORD RESOURCE CONTEXT_KEYWORD DATE SYMBOL
          | ACTION_KEYWORD NUMBER TICKET_TYPE RESOURCE CONTEXT_KEYWORD RESOURCE CONTEXT_KEYWORD DATE SYMBOL
```

LIST_KEYWORD → ‘List all’ | ‘List’

PAYMENT_COMMAND → ACTION_KEYWORD RESOURCE CONTEXT_KEYWORD SERVICE CONTEXT_KEYWORD USERNAME SYMBOL
| PAYMENT_TYPE SYMBOL

CANCELLATION_COMMAND → ACTION_KEYWORD RESOURCE LOCATION_MARKER SERVICE LOCATION_MARKER START_DATE LOCATION_MARKER END_DATE CONTEXT_KEYWORD USERNAME SYMBOL

RENT_COMMAND → RENT_KEYWORD RESOURCE LOCATION_MARKER LOCATION LOCATION_MARKER START_DATE LOCATION_MARKER END_DATE CONTEXT_KEYWORD USERNAME SYMBOL'"

INQUIRY_COMMAND → ACTION_KEYWORD RESOURCE CONTEXT_KEYWORD LOCATION_MARKER DEPARTURE LOCATION_MARKER ARRIVAL SYMBOL

ACTION_KEYWORD → 'Book a'| 'Confirm a'| 'Pay'| 'Cancel a'| 'Reserve a'| 'How many'| 'Duration of'

CONTEXT_KEYWORD → 'on'| 'For'| 'Schedule'| 'are there'| 'Returning'| 'cost' | 'available'

RENT_KEYWORD → 'Rent a' | 'Rent' | 'Rental'

LOCATION_MARKER → 'in'| 'at'| 'from'| 'to'

CONNECTIVE_WORD → 'that'

ARTICLE_CONJUNCTION → 'a' | 'and'

PAYMENT_TYPE → 'credit card' |'debit card' | 'bank transfer'

RESOURCE → 'Reservations' | 'Reservation' | 'Tickets' | 'Ticket' | 'tickets' | 'Flights' | 'Flight' | 'Rooms' | 'Room' | 'Hotels' | 'Hotel'

CONDITIONS → 'less than' | 'more than' | 'equal to' | 'greater than' | 'if' | 'then'

DATE → 'Jan' NUMBER, NUMBER | 'Feb' NUMBER, NUMBER | 'Mar' NUMBER, NUMBER | 'Apr' NUMBER, NUMBER | 'May' NUMBER, NUMBER | 'Jun' NUMBER, NUMBER | 'Jul' NUMBER, NUMBER | 'Aug' NUMBER, NUMBER | 'Sep' NUMBER, NUMBER | 'Oct' NUMBER, NUMBER | 'Nov' NUMBER, NUMBER | 'Dec' NUMBER, NUMBER

START_DATE → DATE

PAYMENT → ‘credit card’ | ‘debit card’ | ‘bank transfer’

CONFIRM_KEYWORD → ‘Confirm a’ | ‘Confirm the’ | ‘Confirm’

INQUIRY_KEYWORD → ‘How many’ | ‘What is the’

PASSENGER_TYPE → ‘adults’ | ‘children’ | ‘seniors’ | ‘students’ | ‘adult’ | ‘child’ | ‘senior’ | ‘student’

END_DATE → DATE

TIME → NUMBER:NUMBER 'AM' | NUMBER:NUMBER 'PM' | NUMBER:NUMBER

NUMBER → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

MONEY → '$' NUMBER | '$' NUMBER.NUMBER

USERNAME → CHAR CHAR CHAR'_'CHAR CHAR CHAR

CHAR → 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'

SYMBOL → '.' | '?'

- **Complete parse tree/AST for a sample program in your language.**

  sample_input_data = “Book a Ticket from Montego Bay to Miami on February 17, 2025 at 8:30 AM returning on March 17, 2025 at 8:30 AM.

  Parsed Result:

  ('COMMAND', ('BOOKING_COMMAND', ('ACTION_KEYWORD', 'Book a'), ('RESOURCE', 'Ticket'), ('LOCATION_MARKER', 'from'), ('DEPARTURE', 'Montego Bay'), ('LOCATION_MARKER', 'to'), ('ARRIVAL', 'Miami'), ('CONTEXT_KEYWORD', 'on'), ('START_DATE', 'Feb 17, 2025'), ('LOCATION_MARKER', 'at'), ('TIME', '8:30 AM'), ('CONTEXT_KEYWORD', 'returning'), ('CONTEXT_KEYWORD', 'on'), ('END_DATE', 'Mar 17, 2025'), ('LOCATION_MARKER', 'at'), ('TIME', '8:30 AM'), ('SYMBOL', '.')))

- **Full list of tokens for the language you developed.**
  - **Tokens:**
    - ACTION_KEYWORD
    - CONTEXT_KEYWORD
    - LOCATION_MARKER
    - CONNECTIVE_WORD
    - DATE
    - START_DATE
    - END_DATE
    - NUMBER
    - SYMBOL
    - MONEY
    - RESOURCE
    - CONDITIONS
    - TIME
    - USERNAME
    - DEPARTURE
    - ARRIVAL
    - LOCATION
    - SERVICE
    - ARTICLE_CONJUNCTION
    - PAYMENT_TYPE
    - INQUIRY_KEYWORD
    - PASSENGER_TYPE
    - CONFIRM_KEYWORD
    - TICKET_TYPE

- **Regular expressions you used to recognize all the tokens for the language you developed.**

  Regular Expressions Used:
  - ACTION_KEYWORD
    - Array specified in reg ex: action_keywords = ['Book a', 'Book', 'Pay', 'Cancel a', 'Reserve a', 'How many', 'Duration of', 'Booking']
    - t_ACTION_KEYWORD = r'\b(?:' + r'|'.join(action_keywords) + r')\b'
  - CONTEXT_KEYWORD
    - Array specified in reg ex: context_keywords = ['on', 'For', 'Schedule', 'are there', 'Returning', 'cost', 'available']
    - t_CONTEXT_KEYWORD = r'\b(?:' + r'|'.join(context_keywords) + r')\b'
  - LOCATION_MARKER
    - Array specified in reg ex: location_markers = ['in', 'at', 'from', 'to']
    - t_LOCATION_MARKER = r'\b(?:' + r'|'.join(location_markers) + r')\b'

- CONNECTIVE_WORD
    - Array specified in reg ex : connective_words = ['that']
    - t_CONNECTIVE_WORD = r'\b(?:' + r'|'.join(connective_words) + r')\b'
- DATE
    - t_DATE = r'(?<=\bon\s)((?!\b(?:in|at|from|to)\b).)+?(?=\.)'
- START_DATE
    - t_START_DATE = r'(?<=\bfrom\b\s).+?(?=\s\bto\b)|(?<=\bon\b\s).+?(?=\s\bat\b)'
- END_DATE
    - t_END_DATE = r'(?<=\breturning on\s).+?(?=\s(?:at)\b)|' \
      r'(?<=\bto\s).+?(?=\s(?:for)\b)|' \
      r'(?<=\bto\s).+?(?=\.)'
- NUMBER
    - t_NUMBER = r'\b\d+\b'
- SYMBOL
    - t_SYMBOL = r'\.+(?=[\t]*$)|,|:'
- MONEY
    - t_MONEY = r'\$\d+(\.\d+)?'
- RESOURCE
    - t_RESOURCE = (
      r'(?<=\bRent a\s|Rental\s|Book a\s)([A-Za-z]+)(?=\sin)|'
      r'Reservations|Reservation|Tickets|Ticket|tickets|Flights|Flight|Rooms|Room|Hotels|Hotel|'
      r'(?<=\bfor\s)([A-Za-z]+(?:\s[A-Za-z]+)?)(?=\s\bon\b)'
)
- CONDITION
    - t_CONDITIONS = r'\b(?:less than|more than|equal to|greater than|if|then)\b'
- TIME
    - t_TIME = r'\b(?:([0-9])?[0-9]):[0-9][0-9]\s*(?:AM|PM)\b'
- USERNAME
    - t_USERNAME = r'(?<=\bfor\b\s)[A-Za-z0-9_]+(?=\.)'
- DEPARTURE
    - t_DEPARTURE = r'(?<=\bfrom\b\s)([a-zA-Z\s]+?)(?=\s\band\b)|(?<=\bFrom\b\s)([a-zA-Z\s]+)(?=\s\bTo\b)|(?<=\bFrom\b\s)([a-zA-Z\s]+)(?=\s\bThat\b)|(?<=\bFrom\b\s)([a-zA-Z\s]+)(?=\s*\.)|(?<=\bFrom\b\s)([a-zA-Z\s]+)(?=\s\b(?:' + r'|'.join(all_keywords) + r')\b)'
- ARRIVAL
    - t_ARRIVAL = r'(?<=\bTo\b\s)([a-zA-Z\s]+?)(?=\s\bFrom\b)|'\
      r'(?<=\bTo\b\s)([a-zA-Z\s]+?)(?=\s*\.)|'\
      r'(?<=\bTo\b\s)([a-zA-Z\s]+?)(?=\s\b(?:' + r'|'.join(all_keywords) + r')\b)'
- LOCATION
    - t_LOCATION = r'(?<=\bin\b\s)([a-zA-Z\s]+?)(?=\s\b(?:' + r'|'.join(all_keywords) + r')\b)|' \
      r'(?<=\bin\b\s)([a-zA-Z\s]+?)(?=\s*\.)|' \
      r'(?<=\bin\s)([a-zA-Z\s]+?)(?=\s\bfrom\b)'

- SERVICE
  - t_SERVICE =
    ```
    r'(?<=\ba\s)(?!(?:' + r'|'.join(all_keywords) + r')\b)([A-Za-z]+(?:\s[A-Za-z]+)?)(?=\s(?:' + t_RESOURCE + r')\b)|' \
    r'(?<=\bList\s)([A-Za-z]+(?:\s[A-Za-z]+)?)(?=\s\bSchedule\b)|' \
    r'(?<=\bList all\s)([A-Za-z]+(?:\s[A-Za-z]+)?)(?=\s\bSchedule\b)|' \
    r'(?<=\bfor\s)([A-Za-z]+(?:\s[A-Za-z]+)?)(?=\s\bfor\b)|' \
    r'(?<=\bat\s)([A-Za-z]+(?:\s[A-Za-z]+)?)(?=\s(?:From|from)\b)|'\
    r'(?<=\bfor\s)([A-Za-z]+(?:\s[A-Za-z]+)?)(?=\s(?:From|from)\b)|'\
    r'(?<=\bConfirm the\s)([A-Za-z]+(?:\s[A-Za-z]+)?)(?=\s(?:' + t_ACTION_KEYWORD + r')\b)|'\
    r'(?<=\bConfirm a\s)([A-Za-z]+(?:\s[A-Za-z]+)?)(?=\s(?:' + t_ACTION_KEYWORD + r')\b)|'\
    r'(?<=\bConfirm\s)([A-Za-z]+(?:\s[A-Za-z]+)?)(?=\s(?:' + t_ACTION_KEYWORD + r')\b)'
    ```
- ARTICLE_CONJUNCTION
  - t_ARTICLE_CONJUNCTION = r'\b(a|and)\b'
- PAYMENT
  - t_PAYMENT_TYPE = r'\b(credit card|debit card|bank transfer)\b'
- CONFIRM_KEYWORD
  - t_CONFIRM_KEYWORD = r'\b(Confirm a|Confirm the|Confirm)\b'
- INQUIRY_KEYWORD
  - t_INQUIRY_KEYWORD = r'\b(How many|What is the)\b'
- PASSENGER_TYPE
  - t_PASSENGER_TYPE=r'\b(adults|children|seniors|students|adult|child|senior|student)\b'
- TICKET_TYPE
  - t_TICKET_TYPE = r'(?<=\d\s)(.*?)(?=\s(?:ticket|tickets))'

- **Demonstration of scope and binding in sample code written in your programming language.**

User Scope and Binding

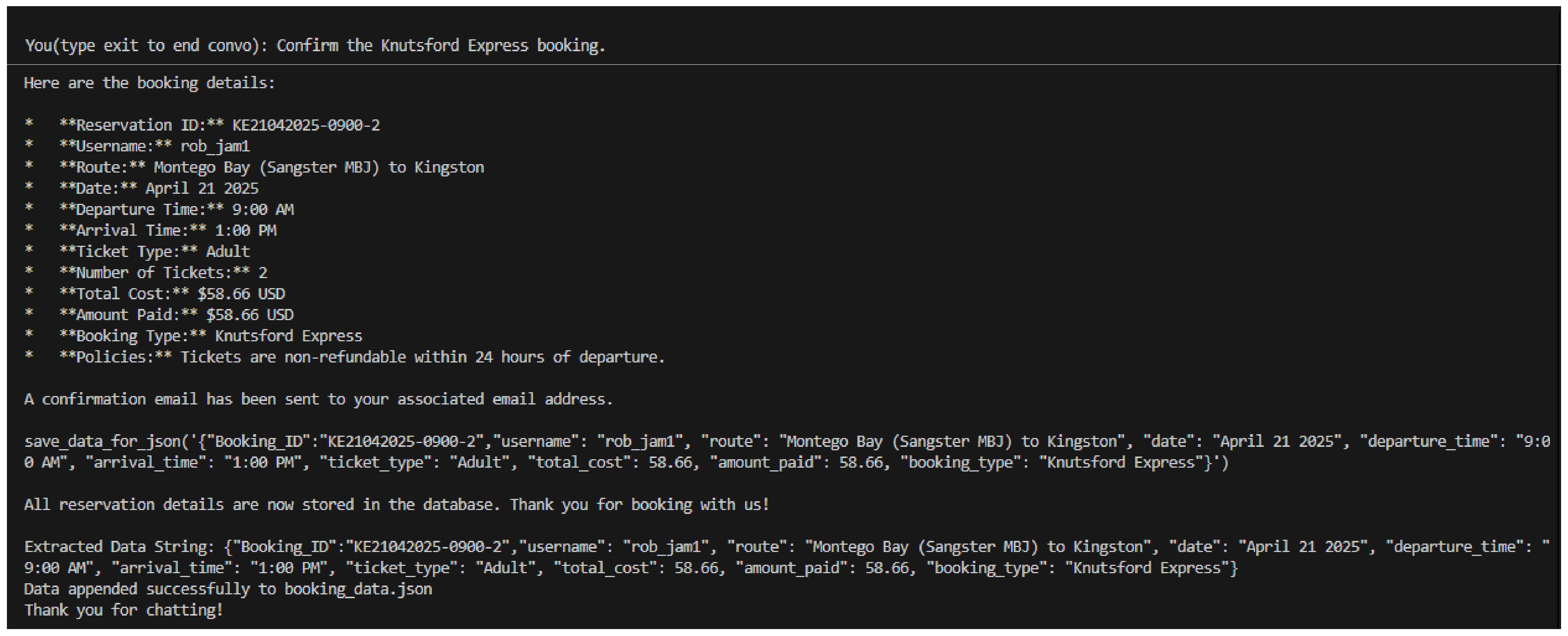
Our booking language implements a user context to maintain information associated with the current user throughout a session. When a user is identified by providing a username, this information is bound to a user context.

This binding ensures that subsequent actions are correctly performed on behalf of that specific user.

For instance:

- Scenario:
  - The user initiates a booking by providing their username: Confirm the Knutsford Express booking for rob_jam1.
  - The system establishes a binding between the user and 'rob_jam1'.
  - Later, the user issues another confirmation: Confirm the Knutsford Express booking.
  - In this subsequent prompt, the username 'rob_jam1' does not need to be explicitly repeated because it remains bound to the user context. The system implicitly operates within the scope of that user's context.

This approach demonstrates how our language manages scope by maintaining the user context. The user identifier retains its binding within a defined scope, allowing actions to correctly reference the associated user.

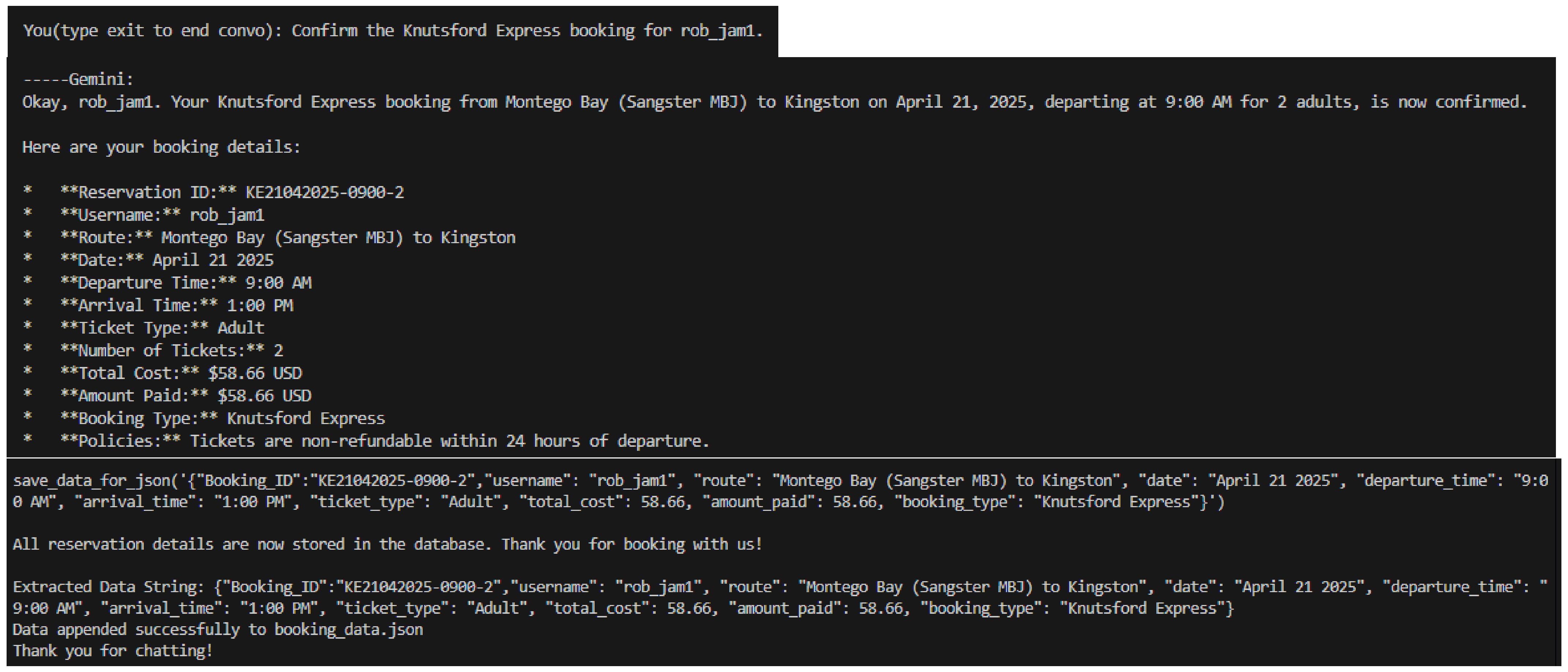


```
You(type exit to end convo): Confirm the Knutsford Express booking.
Here are the booking details:

*   **Reservation ID:** KE21042025-0900-2
*   **Username:** rob_jam1
*   **Route:** Montego Bay (Sangster MBJ) to Kingston
*   **Date:** April 21 2025
*   **Departure Time:** 9:00 AM
*   **Arrival Time:** 1:00 PM
*   **Ticket Type:** Adult
*   **Number of Tickets:** 2
*   **Total Cost:** $58.66 USD
*   **Amount Paid:** $58.66 USD
*   **Booking Type:** Knutsford Express
*   **Policies:** Tickets are non-refundable within 24 hours of departure.

A confirmation email has been sent to your associated email address.

save_data_for_json('{"Booking_ID":"KE21042025-0900-2","username": "rob_jam1", "route": "Montego Bay (Sangster MBJ) to Kingston", "date": "April 21 2025", "departure_time": "9:0
0 AM", "arrival_time": "1:00 PM", "ticket_type": "Adult", "total_cost": 58.66, "amount_paid": 58.66, "booking_type": "Knutsford Express"}')

All reservation details are now stored in the database. Thank you for booking with us!

Extracted Data String: {"Booking_ID":"KE21042025-0900-2","username": "rob_jam1", "route": "Montego Bay (Sangster MBJ) to Kingston", "date": "April 21 2025", "departure_time": "
9:00 AM", "arrival_time": "1:00 PM", "ticket_type": "Adult", "total_cost": 58.66, "amount_paid": 58.66, "booking_type": "Knutsford Express"}
Data appended successfully to booking_data.json
Thank you for chatting!
```

Reservation Scope and Data Binding

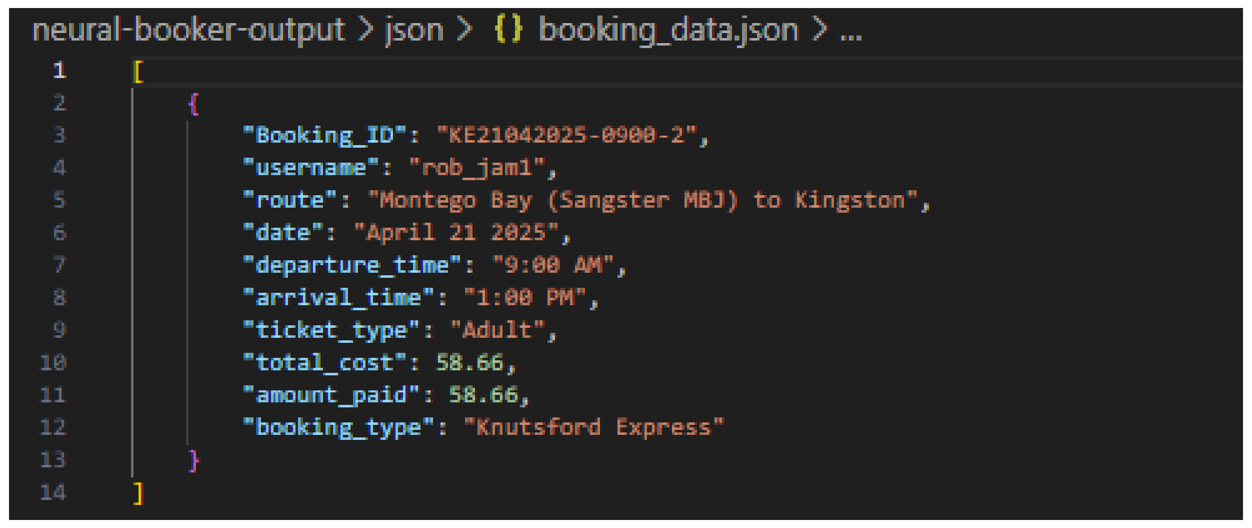
Each booking within our language is associated with a unique reservation context. This context is established when a booking is confirmed and is crucial for maintaining the integrity of booking-specific data. A key aspect of this context is the binding of a unique reservation ID to a collection of data representing the booking details.

The save_data_for_json() function exemplifies this binding. When the AI want to store information it outputs save_data_for_json(…..). There is some code that identifies when the AI responds says the trigger word save_data_for_json and it takes the data in the parameter and creates a structured data representation (a JSON object) where specific booking details (e.g., Booking_ID, username, route, etc.) are bound to corresponding keys. This JSON object then effectively acts as the scope for all information pertaining to that particular reservation.



```
You(type exit to end convo): Confirm the Knutsford Express booking for rob_jam1.

-----Gemini:
Okay, rob_jam1. Your Knutsford Express booking from Montego Bay (Sangster MBJ) to Kingston on April 21, 2025, departing at 9:00 AM for 2 adults, is now confirmed.

Here are your booking details:

*   **Reservation ID:** KE21042025-0900-2
*   **Username:** rob_jam1
*   **Route:** Montego Bay (Sangster MBJ) to Kingston
*   **Date:** April 21 2025
*   **Departure Time:** 9:00 AM
*   **Arrival Time:** 1:00 PM
*   **Ticket Type:** Adult
*   **Number of Tickets:** 2
*   **Total Cost:** $58.66 USD
*   **Amount Paid:** $58.66 USD
*   **Booking Type:** Knutsford Express
*   **Policies:** Tickets are non-refundable within 24 hours of departure.
save_data_for_json('{"Booking_ID":"KE21042025-0900-2","username": "rob_jam1", "route": "Montego Bay (Sangster MBJ) to Kingston", "date": "April 21 2025", "departure_time": "9:0
0 AM", "arrival_time": "1:00 PM", "ticket_type": "Adult", "total_cost": 58.66, "amount_paid": 58.66, "booking_type": "Knutsford Express"}')

All reservation details are now stored in the database. Thank you for booking with us!

Extracted Data String: {"Booking_ID":"KE21042025-0900-2","username": "rob_jam1", "route": "Montego Bay (Sangster MBJ) to Kingston", "date": "April 21 2025", "departure_time": "
9:00 AM", "arrival_time": "1:00 PM", "ticket_type": "Adult", "total_cost": 58.66, "amount_paid": 58.66, "booking_type": "Knutsford Express"}
Data appended successfully to booking_data.json
Thank you for chatting!
```

Data saved in a JSON file:

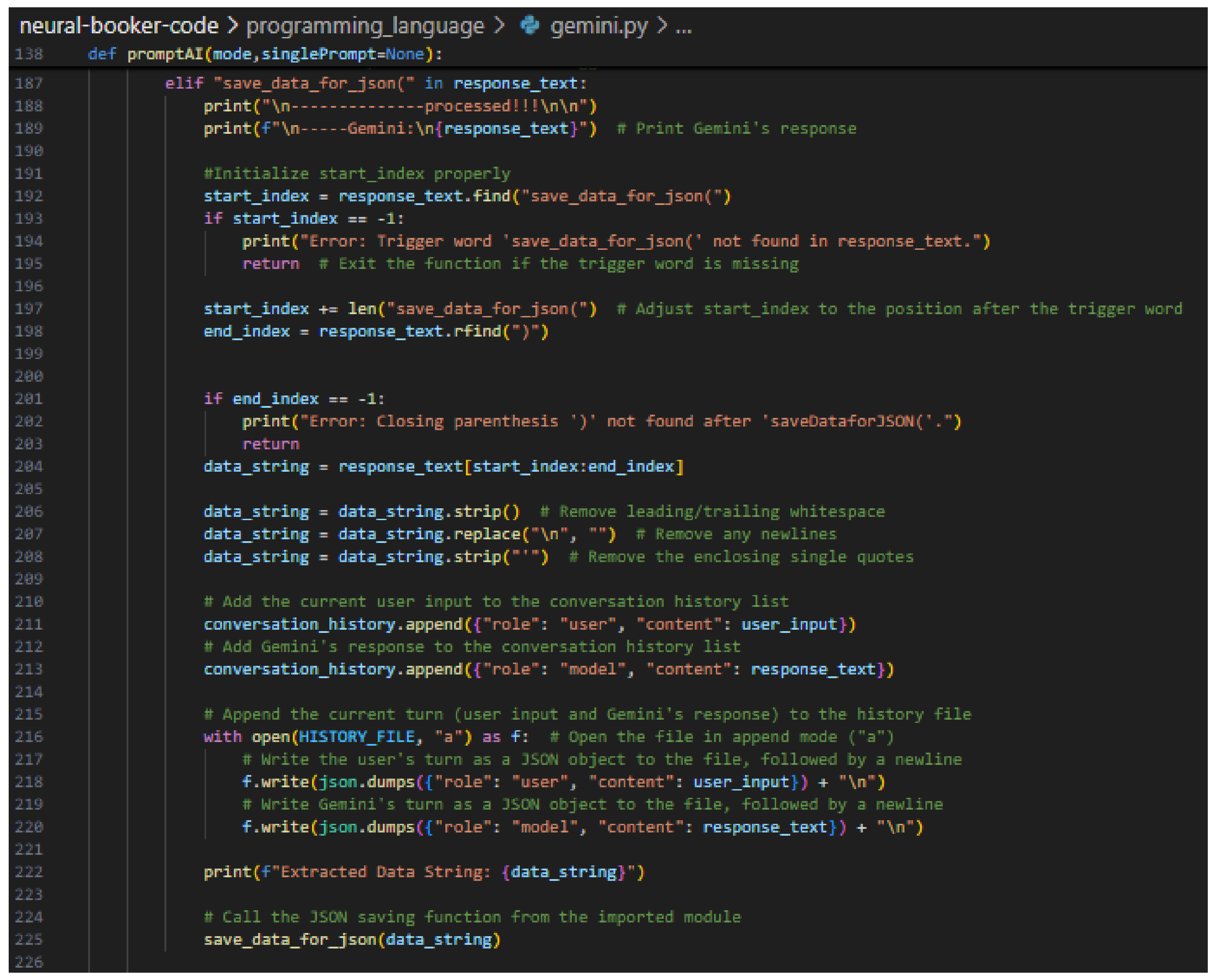


neural-booker-output > json > {} booking_data.json > ...

```json
[
    {
        "Booking_ID": "KE21042025-0900-2",
        "username": "rob_jam1",
        "route": "Montego Bay (Sangster MBJ) to Kingston",
        "date": "April 21 2025",
        "departure_time": "9:00 AM",
        "arrival_time": "1:00 PM",
        "ticket_type": "Adult",
        "total_cost": 58.66,
        "amount_paid": 58.66,
        "booking_type": "Knutsford Express"
    }
]
```

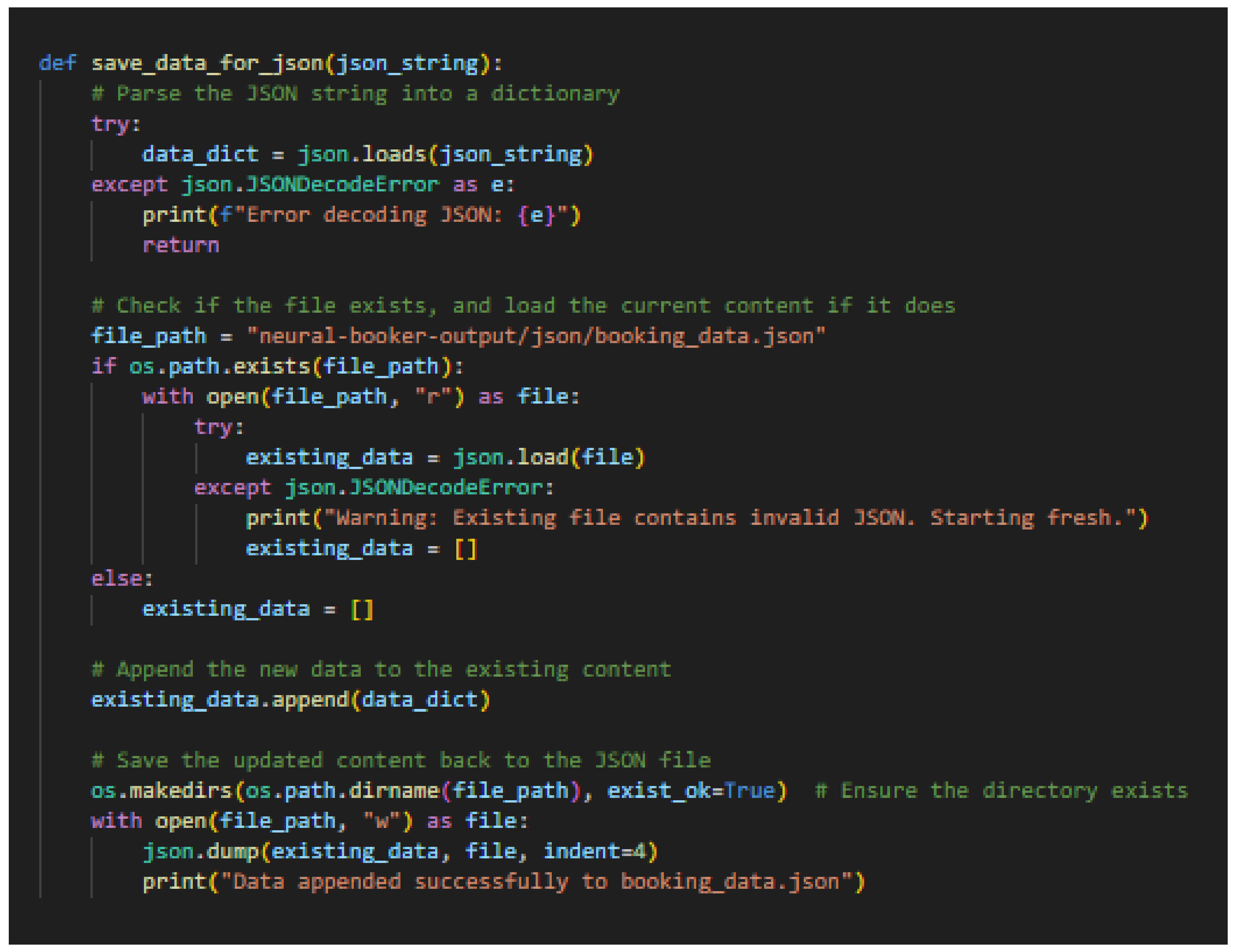
- save_data_for_json('{"Booking_ID":"KE21042025-0900-2","username": "rob_jam1", ...}')
- Here, keys such as Booking_ID, username, etc., are bound to their respective values. This JSON object becomes the scope for all information related to reservation 'KE21042025-0900-2'.
- This data can further be used to retrieving, modifying, or canceling the booking.

Code that tracks AI response checking for trigger word



neural-booker-code > programming_language > gemini.py > ...

```python
    def promptAI(mode,singlePrompt=None):
            elif "save_data_for_json(" in response_text:
                print("\n--------------processed!!!\n\n")
                print(f"\n-----Gemini:\n{response_text}")  # Print Gemini's response

                #Initialize start_index properly
                start_index = response_text.find("save_data_for_json(")
                if start_index == -1:
                    print("Error: Trigger word 'save_data_for_json(' not found in response_text.")
                    return  # Exit the function if the trigger word is missing

                start_index += len("save_data_for_json(")  # Adjust start_index to the position after the trigger word
                end_index = response_text.rfind(")")


                if end_index == -1:
                    print("Error: Closing parenthesis ')' not found after 'saveDataforJSON('.")
                    return
                data_string = response_text[start_index:end_index]

                data_string = data_string.strip()  # Remove leading/trailing whitespace
                data_string = data_string.replace("\n", "")  # Remove any newlines
                data_string = data_string.strip("'")  # Remove the enclosing single quotes

                # Add the current user input to the conversation history list
                conversation_history.append({"role": "user", "content": user_input})
                # Add Gemini's response to the conversation history list
                conversation_history.append({"role": "model", "content": response_text})

                # Append the current turn (user input and Gemini's response) to the history file
                with open(HISTORY_FILE, "a") as f:  # Open the file in append mode ("a")
                    # Write the user's turn as a JSON object to the file, followed by a newline
                    f.write(json.dumps({"role": "user", "content": user_input}) + "\n")
                    # Write Gemini's turn as a JSON object to the file, followed by a newline
                    f.write(json.dumps({"role": "model", "content": response_text}) + "\n")

                print(f"Extracted Data String: {data_string}")

                # Call the JSON saving function from the imported module
                save_data_for_json(data_string)
```



```python
def save_data_for_json(json_string):
    # Parse the JSON string into a dictionary
    try:
        data_dict = json.loads(json_string)
    except json.JSONDecodeError as e:
        print(f"Error decoding JSON: {e}")
        return

    # Check if the file exists, and load the current content if it does
    file_path = "neural-booker-output/json/booking_data.json"
    if os.path.exists(file_path):
        with open(file_path, "r") as file:
            try:
                existing_data = json.load(file)
            except json.JSONDecodeError:
                print("Warning: Existing file contains invalid JSON. Starting fresh.")
                existing_data = []
    else:
        existing_data = []

    # Append the new data to the existing content
    existing_data.append(data_dict)

    # Save the updated content back to the JSON file
    os.makedirs(os.path.dirname(file_path), exist_ok=True)  # Ensure the directory exists
    with open(file_path, "w") as file:
        json.dump(existing_data, file, indent=4)
        print("Data appended successfully to booking_data.json")
```

- **Details on the programming language you used to develop your compiler.**

  The compiler for the developed programming language was implemented using Python, a high-level, general-purpose programming language known for its readability, simplicity, and extensive standard library. Python was chosen due to its strong support for rapid development, ease of syntax, and availability of robust third-party tools for language processing. To handle the lexical analysis and parsing phases of the compiler, the PLY (Python Lex-Yacc) library was utilized. PLY provides implementations of lexers (lex) and parsers (yacc) similar to the traditional Lex and Yacc tools found in C, but entirely written in Python. This allowed for efficient tokenization and syntax analysis while maintaining full integration within the Python ecosystem. The combination of Python and PLY enabled a clean, modular design for the compiler and significantly accelerated development by abstracting away many of the lower-level details typically associated with compiler construction.

- **Two characteristics of a good programming language (from those you studied in class) that are evident in your designed programming language, and examples of how do these characteristics affect the readability, writability and reliability of your designed programming language.**

  Syntax Design

  Natural language-like syntax allows users to interact with a programming language in a way that closely resembles everyday human communication. This enhances accessibility and makes the language intuitive, especially for users without extensive programming experience.

  Example in Our Design

  Our language is designed to enable users to issue commands naturally, as if they were speaking or writing in regular conversation. This is achieved by supporting flexible, human-readable constructs, such as:

  ▪ Flexible Phrasing:

    Users can express similar commands in multiple natural ways without losing functionality:

    ○ List all Knutsford Express schedules from Kingston to Montego Bay.

    ○ What is the schedule for Knutsford Express from Kingston to Montego Bay on Apr 21, 2025?"`

    Both commands are tokenized and parsed correctly, allowing seamless interaction.

Impact on Readability, Writability, and Reliability

- Readability:
  - Commands mimic natural conversation, making it simple for users to understand the intent of a command at a glance.
  - For example, What is the schedule for Knutsford Express? is inherently more intuitive than a highly technical query format.

- Writability:
  - Users don’t need to memorize complex syntax rules or specific conventions. They can write commands naturally, reducing the learning curve and improving productivity.

- Reliability:
  - By supporting natural syntax and case insensitivity, the language reduces the likelihood of syntax errors, ensuring commands are processed accurately even if users deviate slightly from standard phrasing.

Simplicity

Simplicity in a programming language refers to the ease with which users can learn, write, and understand commands. A simple language minimizes unnecessary complexity, allowing users to focus on their tasks without distraction.

Example in Our Design

Our language is case insensitive, meaning users can write commands in any case fully lowercase, uppercase, or a mix and the lexer will tokenize them correctly. This reduces the cognitive load on users, making the language easier to work with.

Command in Mixed Case: “List all Knutsford Express schedule”

Command in Mixed Case: “list all knutsford express schedule”

Command in Mixed Case: “LIST ALL KNUTSFORD EXPRESS SCHEDULE”

All these variations produce the same tokenized output, ensuring that users can work naturally without worrying about letter casing.

Impact on Readability, Writability, and Reliability

- Readability:
  - Commands are easy to read, regardless of how the user chooses to capitalize text.

- Writability:
  - Users are not burdened with strict capitalization rules, allowing for flexible and intuitive input

- Reliability:
  - By normalizing case sensitivity, the language avoids errors caused by mismatched casing, ensuring consistent interpretation of commands.

Contributions

Name	**Contribution**
Kemar Christie	Creation of Lexer & Parser, Database & Documentation
Roberto James	Creation of Lexer & Parser, Training of AI & Documentation
Dwayne Gibbs	Creation of Lexer & Parser, Scraping Tools Implementation & Documentation
Tyoni Davis	Creation of Lexer & Parser, Database & Documentation
Danielle Jones	Creation of Lexer & Parser, Training of AI & Documentation