ADVANCED PROGRAMMING

C-CS219

Spring 2025

Astronomy chatbot

Team Members

Student Name	Student ID
Maroska Osama	23-101209
Dania Hassan	23-101147
Jana Abdelmoniem	23-101148
Mohamed Mostafa	24-101222

Table of contents

1. Abstract
2. Introduction
3. Motivation
4. System design
    - System modules and functions
    - Hierarchy chart
5. Description
6. Team's workload distribution
7. User manual
8. Testing and validation
9. Challenges and conclusion

References

The abstract:

In a universe full of mysteries, what if you could explore the wonders of space through a friendly, intelligent guide? This project introduces an interactive astronomy chatbot developed in Scala, designed to simulate conversation, deliver space-related facts, and assess user knowledge through quizzes—all while showcasing the power of functional programming. Leveraging pattern matching, immutable data structures, and pure functions, the chatbot provides a seamless educational experience. Whether you're curious about black holes or eager to test your knowledge of the solar system, this chatbot blends learning with conversation, bringing the cosmos a little closer—one message at a time.

The introduction:

What if learning about the universe felt more like chatting with a friend than reading a textbook? That’s exactly what we set out to build with our **Astronomy Chatbot**—a conversational assistant powered by Scala and functional programming principles. Designed for both **terminal** and **GUI using python**, the chatbot delivers an engaging, interactive experience that makes space science accessible and enjoyable.

Our chatbot doesn’t just respond to commands—it **holds friendly conversations**, shares **random and fascinating astronomy facts**, and offers two styles of quizzes: a **traditional knowledge-based quiz** to test what you know, and a **personality-style quiz** that analyzes your answers to give you a space-themed result (e.g., “Which celestial object are you?”).

From black holes to supernovas, the chatbot makes exploring the cosmos feel less like a lecture and more like a journey—whether you're curious, competitive, or just killing time under the stars.

What made us choose the astronomy chatbot?

🔭 **A love for the stars**: Space is vast, beautiful, and mysterious. We wanted to share that excitement with others in a way that’s easy to access and enjoy.

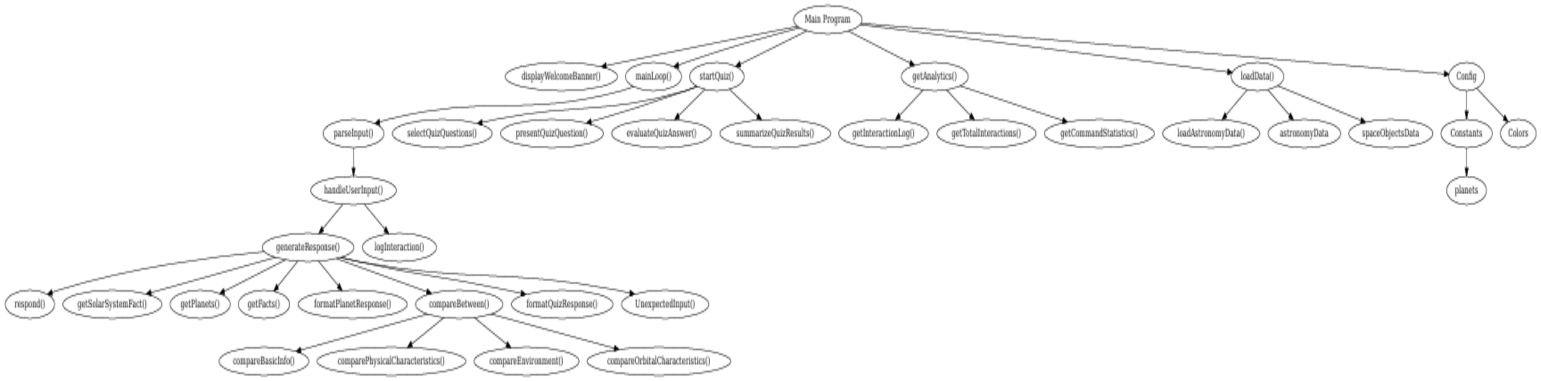
□ **Combining education with interaction**: We aimed to create more than a bot that answers questions. We envisioned a learning companion that talks, quizzes, and responds like a curious friend.

💬 **Making conversations meaningful**: Most chatbots feel robotic. Ours is different—it tells random fun facts, answers intelligently, and keeps users engaged with a natural tone.

□ **Showcasing Scala’s potential**: We wanted to prove that Scala, often seen as complex, can power friendly and functional applications using features like pattern matching, immutable structures, and pure functions.

🌐 **Reaching users everywhere**: With both terminal and web versions, we ensured our chatbot is flexible and accessible no matter how users choose to interact.

the hierarchy chart:



The functions used in our program:

1. **Main.main** - Program entry point that initializes all components and launches the chatbot.
2. **mainLoop** - Primary execution loop for processing user input and displaying responses.
3. **initializeDataSource** - Sets up astronomy data sources and creates sample data if needed.
4. **createSampleDataFile** - Creates a basic astronomy data file when none exists.
5. **promptUserName** - Requests and captures the user's name for personalized interactions.
6. **displayWelcomeBanner** - Shows ASCII art and welcome message at startup.
7. **QuizManager.apply** - Factory method that creates a new QuizManagerImpl instance.
8. **QuizManagerImpl.handleMessage** - Processes user input during quiz mode.
9. **QuizManagerImpl.isQuizActive** - Checks if a quiz is currently running.
10. **QuizManagerImpl.resetQuizState** - Resets the quiz state after completion.
11. **InputParser.parseInput** - Converts user text into structured commands.
12. **ResponseGenerator.respond** - Main response generation function based on commands.
13. **ResponseGenerator.getSolarSystemFact** - Gets random solar system facts.
14. **ResponseGenerator.getPlanets** - Returns list of planets in the solar system.
15. **ResponseGenerator.getFacts** - Retrieves facts about a specific astronomy topic.
16. **ResponseGenerator.formatPlanetResponse** - Creates formatted planet information.
17. **ResponseGenerator.formatCategoryResponse** - Formats responses about astronomy categories.
18. **ResponseGenerator.getRandomUnknownResponse** - Generates responses for unrecognized inputs.
19. **ResponseGenerator.getRandomGreeting** - Provides random greeting messages.
20. **ResponseGenerator.getHelpMessage** - Returns help text showing available commands.
21. **ResponseGenerator.greetUser** - Creates personalized user greeting.
22. **ResponseGenerator.formatQuizResponse** - Formats quiz output with colors/styling.
23. **ResponseGenerator.compareBetween** - Compares two astronomical objects.

24. **ResponseGenerator.compareProperties** - Helper method for comparing object properties.
25. **ResponseGenerator.generateUniqueFeatures** - Identifies unique features between objects.
26. **ResponseGenerator.formatComparisonResponse** - Formats object comparison output.
27. **ResponseGenerator.UnexpectedInput** - Handles special case inputs with appropriate responses.
28. **ResponseGenerator.compareBasicInfo** - Compares basic information between objects.
29. **ResponseGenerator.comparePhysicalCharacteristics** - Compares physical traits between objects.
30. **ResponseGenerator.compareEnvironment** - Compares environmental conditions between objects.
31. **ResponseGenerator.compareOrbitalCharacteristics** - Compares orbital properties between objects.
32. **AnalyticsImpl.logInteraction** - Records user command patterns.
33. **AnalyticsImpl.getTotalInteractions** - Returns total number of user interactions.
34. **AnalyticsImpl.getMostFrequentCommand** - Identifies most commonly used command.
35. **AnalyticsImpl.getCommandStatistics** - Returns statistics on all command usage.
36. **DataLoader.loadAstronomyData** - Loads astronomy data from JSON files.

The description of our code contains the explanation:

Function Name / Value	**Purpose**	**Detailed Explanation for object constants**
CMD_* constants	Store predefined command identifiers	Used to standardize chatbot commands like HELP, START_QUIZ, etc., so different modules can reference them consistently.
helpWords, listWords, etc.	Word lists for intent matching	These are lists of synonyms and trigger phrases used by the chatbot to detect what the user is trying to say (e.g., asking for help or requesting a list of planets).
planets and categories	Domain-specific keywords	These are lists of actual celestial body names and categories that the chatbot recognizes and understands.

Function Name / Value	Purpose	Detailed Explanation for object colors
Reset, Red, Green, etc.	ANSI escape codes for terminal text formatting	Used to print colored or styled text in the terminal, useful for emphasizing errors or important messages (e.g., red for warnings).

Function Name / Value	**Purpose**	**Detailed Explanation for object config**
load: Config	Load configuration object	Currently returns a default instance of Config. Placeholder for future enhancements like loading settings from a file.

Function Name / value	Purpose	Detailed Explanation for object data loader
loadAstronomyData(filename)	Load JSON astronomy data from a file	Parses a JSON file containing an array of celestial objects. It manually splits and converts each object string into a Map[String, String], indexed by lowercase object names.
splitObjects(str)	Helper to split raw JSON array into objects	Iterates through characters while tracking braces and quotes to accurately extract JSON object strings.
parseObject(objectStr)	Convert a JSON object string to key-value map	Parses individual object strings into a Map, cleaning quotes and commas.
splitKeyValuePairs(str)	Break a JSON object into key-value strings	Tracks brackets and quotes to safely split on commas that separate fields.
loadSpaceObjectsData(filename)	Load CSV-based space object data	Parses a CSV file into a map from object names to field maps. It handles quoted commas and malformed lines.
mergeData(jsonData, csvData)	Combine JSON and CSV data	Merges the two sources. If an object is in both, the CSV fields override the JSON ones.
astronomyData (lazy val)	Lazy-loaded merged astronomy dataset	Combines parsed JSON and CSV files using mergeData.
spaceObjectsData (lazy val)	Lazy-loaded space object dataset from CSV	Loads CSV file only once on demand.

Function Name	Purpose	Detailed Explanation for object InputParser
parseInput(input, isQuizActive)	Main input dispatcher	Normalizes input and routes to quiz mode or regular mode. Translates input into an internal command string like startquiz or compare_earth_mars.
containsAny(words, targetList)	Check if input contains any keyword	Returns true if at least one word in words matches an item in targetList.
matchesGreetings(words)	Detect greeting input	Uses containsAny to match against greeting keywords.
matchesHelp(words)	Detect help-related input	Identifies if user is asking for help or instructions.
matchesListPlanets(words)	Detect requests to list planets	Matches combinations of list words and "planets".
matchesRandomFact(words)	Detect random fact request	Flexible matching for various ways of asking for a fact.
matchesQuiz(words)	Detect quiz intent	Matches quiz-related words like "test", "challenge".

Function Name	Purpose	Detailed Explanation for object InputParser
matchesCompare(input)	Detect comparison input	Checks whether comparison-related terms exist in the input string.
matchesCategory(words)	Detect category listing intent	Matches input against known astronomy categories.
matchesPlanet(words)	Detect planet-specific input	Looks for any known planet name in the words.
extractCompareTopics(input)	Extracts two objects to compare	Replaces all comparison-related keywords with delimiters and extracts two non-empty topics.
extractTopic(input)	Cleans user queries to extract topic	Removes prefixes ("tell me about...") and filler words to isolate a clean topic keyword.
parseQuizMode(words, input)	Command parsing during quiz mode	Recognizes commands like "exit quiz", "skip", or considers anything else as an answer.
parseRegularMode(words, input)	Command parsing during regular mode	Detects commands like HELP, LIST_PLANETS, or extracts a topic, falling back to CMD_UNKNOWN if no clear intent is found.

Function/Method	Purpose	Detailed Explanation for quizmanagerimpl
resetQuizState()	Resets the state of the quiz.	This method resets the quizState to None, effectively ending the current quiz session and clearing any saved progress or results.
isQuizActive()	Checks if a quiz is currently active.	This method returns a boolean value indicating whether the quizState is Some (indicating an active quiz session) or None (indicating no active quiz).
selectQuizQuestions(topic: String)	Selects quiz questions based on the given topic.	Based on the input topic, this method returns a list of Question objects. It supports two topics: "traditional" (astronomy-related questions) and "personal" (questions based on user preferences or general knowledge). If an invalid topic is given, it returns a "Topic not found" question.
normalizeNumber(input: String)	Normalizes numeric inputs to remove unwanted characters.	This method removes non-numeric characters (such as letters and symbols) and standardizes the input to a clean number format (e.g., stripping commas, trimming whitespace). It is used for comparing numeric answers in the quiz.
compareNumbers(answer: String, correct: String, tolerance: Double = 0.1)	Compares the user's numeric answer to the correct answer with a tolerance.	This method normalizes the user's answer and the correct answer, then compares them numerically. The comparison allows for a small tolerance (default is 0.1), meaning the answer is considered correct if the difference between the two numbers is within this margin.

Function/Method	**Purpose**	**Detailed Explanation for quizmanagerimpl**
checkSynonyms(userAnswer: String, correctAnswer: String)	Checks if the user's answer contains any synonyms of the correct answer.	This method checks if the user's answer contains any word or synonym from a predefined list of synonyms stored in synonymMap. It returns true if a synonym match is found for both the user’s and correct answer, making the comparison more flexible and forgiving of different terms for the same object.
isNumericAnswer(answer: String)	Determines if the provided answer is numeric.	This method checks whether the provided answer contains any digits, returning true if it's a numeric answer (including decimal numbers) and false otherwise.
presentQuestion(questions: List[Question], index: Int)	Presents a question at the specified index.	This method retrieves the question at the given index from a list of Question objects. If the index is within the bounds of the list, it returns the question with its options; otherwise, it notifies that no more questions are available.
processTopicSelection(message: String)	Processes the user’s topic selection for starting a quiz.	This method takes the user's message as input and checks for valid quiz type selections (e.g., "traditional" or "personal"). It updates the quizState accordingly and returns the first question for the selected topic. If an invalid option is selected, it requests the user to choose a valid quiz type.
levenshteinDistance(s1: String, s2: String)	Calculates the Levenshtein distance between two strings.	This method computes the number of single-character edits (insertions, deletions, substitutions) required to change one string into another. It is used for fuzzy matching when comparing the user’s input to valid options, allowing for minor spelling errors.
findClosestMatch(input: String, options: List[String])	Finds the closest matching option from a list based on Levenshtein distance.	This method takes the user's input and attempts to find the closest match in a list of options by calculating the Levenshtein distance between the input and each option. It returns the closest match if the distance is within a tolerance (distance of 2 or less).
validateQuizAnswer(answer: String, options: List[String])	Validates the user’s answer against a list of options.	This method checks if the user's answer matches one of the valid options (case insensitive). It also checks if the answer is a close match using fuzzy matching (Levenshtein distance). The answer is considered valid if it matches or closely resembles one of the provided options.
handleQuizAnswer(answer: String)	Handles and processes the user’s answer to the current quiz question.	This method checks the validity of the user’s answer. If the answer is valid, it checks if the answer is correct (for both traditional and personal quizzes). The method updates the quiz state (quizState) and progress, providing feedback (correct/incorrect) and advancing to the next question or completing the quiz.

Name	Type	Detailed Explanation for QuizManagerImpl
quizState	Option[QuizState]	This mutable variable holds the user's ongoing quiz session as an Option. When defined (Some), it contains:– selected topic– question list– current index– correct answer count– question results (user answers + correctness).
QuizState	case class	Includes:– topic: String: the quiz topic (e.g., "traditional")– questions: List[Question]: the selected quiz questions– currentIndex: Int: which question the user is currently on– correctCount: Int: how many answers were correct– results: List[(String, Boolean)]: each user answer and whether it was correct
Question	case class	Fields:– text: String: the question itself– options: List[String]: possible answers– correctAnswer: String: the correct option– explanation: Option[String]: optional explanation shown after the user answers.
synonymMap	Map[String, Set[String]]	A mapping from keywords (e.g., "mars") to sets of valid alternatives (e.g., {"the red planet", "planet mars", "mars"}). Used to validate free-form answers by comparing inputs against all synonyms.
traditionalQuestions	List[Question]	Predefined multiple-choice questions focused on scientific topics like planets, distances, and physical properties. Each entry is a Question object with options and explanations.
personalQuestions	List[Question]	A more relaxed question set with personal or imaginative themes (e.g., favorite planet, living on Mars). Designed to be fun or reflective rather than scientifically strict.

Function / Part	Purpose	Description for QuizManager
updateQuizResults	Updates cumulative quiz results for a topic	Takes the previous quiz results, the topic, and new answers. It updates the score and question count if the topic exists or adds a new topic entry.
generateQuizSummary	Generates a summary of the most recent quiz attempt	Returns a formatted string summarizing correct answers, total questions, percentage score, and question review—adapted based on the topic type.
endQuiz	Ends the current quiz session	Finalizes the quiz, clears the quizState, and returns a summary if answers were given, or a message if the quiz was not attempted.
getQuizSummary	Shows cumulative quiz performance	Aggregates and displays total scores and accuracy percentages per quiz topic from all past attempts.

Function / Part	**Purpose**	**Description for QuizManager**
handleMessage	Main message handler for quiz interactions	Determines context (awaiting topic/answer) and dispatches user input to the right function. Also handles commands like "start quiz", "help", etc.
QuizManager	Factory method for creating QuizManagerImpl objects	Allows instantiating the QuizManagerImpl with encapsulated state and logic.

Function / Part	Purpose	Description for QuizManager
updateQuizResults	Update the results of a completed quiz	Updates the stored QuizResults for a given topic by adding the new correct and total answers; if the topic is new, it appends a new entry.
generateQuizSummary	Generate a detailed summary of a completed quiz	Builds a quiz summary string with score and per-question feedback, adjusting formatting for "traditional" and "personal" quiz types.
endQuiz	End the current quiz session	Ends the quiz, generates a summary if answers exist, clears quiz state, and returns an appropriate message.
getQuizSummary	Retrieve a summary of all quiz history	Returns a formatted history of all completed quiz results stored in quizResults.
handleMessage	Main dispatcher for handling user input in quiz mode	Routes the user message based on current quiz state and input content, supporting quiz start, answers, summaries, help, etc.
QuizManager.apply	Factory method to create a quiz manager instance	Instantiates a new QuizManagerImpl object.

Function/Part	**Purpose**	**Detailed Explanation**
comparePhysicalCharacteristics	Compare physical features of two astronomical objects.	Takes two Map[String, String] objects (e.g., for planets). Uses safe lookups with .get(...) to extract "diameter", "mass", and "surface_composition". Builds comparison strings for each attribute. If data exists, returns a formatted string with the comparisons; otherwise returns None.
compareEnvironment	Compare environmental conditions of two objects.	Extracts "temperature", "atmosphere", and "moons" fields using .get(...). Constructs human-readable comparison lines (e.g., atmosphere type or moon count). Returns None if no comparable data is found. Otherwise returns a multi-line string.
compareOrbitalCharacteristics	Compare orbital data like distance from Sun and rotation period.	Safely extracts "distance_from_sun", "orbital_period", and "rotation_period" from each object. Builds formatted comparison strings. Skips fields that are missing in either object. Returns a combined string or None if nothing to compare.
UnexpectedInput	Respond to non-command inputs like jokes,	Cleans the input (trims, lowercases), matches it against a list of predefined responses (e.g., “I love you”, “tell me a joke”), and returns a witty or relevant message. If the input isn’t

Function/Part	Purpose	Detailed Explanation
	emotional queries, or greetings.	recognized, it calls getRandomUnknownResponse() to generate a fallback reply.
respond	Main chatbot routing function.	Receives a command, data, logger, and quiz manager. Handles various commands: unknown_* → UnexpectedInput, help → help message, listplanets/listcategory_* → lists from dataset, randomfact → random fact, askabout_* → specific data lookup, compare_*_* → invokes comparison functions, greetings → greeting response. Central command dispatch logic.
getDataForObject	Retrieve detailed data for a specific object.	Calls getDataForCategory with appropriate category and object key. Retrieves specific data stored in the astronomical_objects object (like "mass", "diameter", etc.).
getDataForCategory	Retrieve all data for a specific category (e.g., planets, stars).	Searches the dataset for a specific category like planets, stars, and returns the list of objects (e.g., names). For each object in the category, retrieves detailed information for use by other functions.
getRandomFact	Retrieve a random fact about astronomical objects.	Selects a random object from a predefined list (e.g., planets, stars) and calls a function like getDataForObject to gather a fact about it. Can include physical characteristics or environmental data.
getRandomUnknownResponse	Return a random default response for unrecognized input.	Randomly selects a message from a predefined list to respond when input does not match known commands or queries.
listPlanets	List all planets in the dataset.	Extracts the names of all planets from the astronomical_objects dataset and formats them into a readable list.
listCategoryPlanets	List all objects in the planets category.	Uses the getDataForCategory function to list all objects in the planets category.
listCategoryStars	List all objects in the stars category.	Uses getDataForCategory to retrieve and list all objects from the stars category.
getQuizQuestions	Fetch a list of quiz questions from a dataset.	Retrieves a collection of questions and their answers for quiz generation.
getScore	Calculate the score of a user's answers in a quiz.	Evaluates the user’s responses to quiz questions, compares them to correct answers, and calculates the score.
updateQuizData	Update quiz progress and data.	Updates quiz-related data in a user’s session, ensuring the correct progress is tracked for each question, answer, and score.

Function/Part	Purpose	Detailed Explanation
addResponseToHistory	Add a user response to the history log for analysis.	Appends each user's response to a history list for tracking. This helps evaluate interactions with the chatbot.
parseCommand	Parse the user input and recognize the command type.	Parses the raw input from the user and identifies the corresponding command type, such as questions about planets, stars, or comparison operations.
getHelpMessage	Provide help information for commands.	Returns a list of available commands along with brief descriptions of what each command does.
compareData	Compare data from two objects.	Uses the compare functions (comparePhysicalCharacteristics, compareEnvironment, compareOrbitalCharacteristics) to compare multiple characteristics of two given astronomical objects.
loadAstronomicalData	Load all astronomical data from a predefined dataset.	Loads the dataset containing information about astronomical objects, including planets, stars, and other relevant categories for the chatbot to use during its interactions.
checkCommandValidity	Check if a command is valid or known.	Verifies whether the user's command exists in the predefined list of available commands. If the command is not recognized, responds with an error message.
getDetailedObjectData	Retrieve detailed information for a specific object.	Based on a recognized object or category, this function gathers a range of detailed data (physical, environmental, orbital) to be used in chatbot responses.
setUpLogger	Set up a logging mechanism for the chatbot.	Configures logging tools to track chatbot interactions, errors, and responses, for debugging and analytics.

The workload distribution:

Team Member	Tasks
Maroska Osama	**Coding Tasks:** - Implement functions for the astronomy chatbot (answer checking, quiz logic, core functions related to quiz processing) - Handle error fixing and code optimization - Implement synonym handling for quiz answers - Address spelling mistakes and handle fuzzy matches in input parsing - Implement and maintain the quiz state management **Documentation & Explanation:** - made the report **Testing & Debugging:** - Test the core quiz functionality, check for bugs and edge cases **Collaboration:** - Assist with integration tasks and provide input for improving input parsing and response handling
Mohamed Mostafa	**Web Server Development:** - Set up and maintain the web server for the astronomy chatbot - Integrate the backend chatbot with the frontend **Backend Development:** - Work on data flow and communication between the chatbot and the web server **Testing & Deployment:** - Conduct tests to ensure proper communication and functionality between the web server and the chatbot - Assist with debugging and deployment - helped in parser input and the main function
Dania Hassan	**Input Parsing:** - Work on parsing user inputs and mapping responses - Handle special cases for user input (e.g., greetings, unclear queries) - Implemented validation functions for responses **Testing & Debugging:**

Team Member	Tasks
	- Test different types of inputs for handling edge cases - Debug and optimize input parsing **Collaboration:** - Work closely with Maroska on testing quiz functionality and answer validation
Jana Abdelmoniem	**Presentation Development:** - Create the PowerPoint presentation, including slides about the chatbot’s features, functions, and purpose **Input Parsing Support:** - Assist Dania in input parser development, particularly for greeting and user interaction functions **Documentation:** - Help with documenting the overall structure and major features of the project **Collaboration:** - Provide input on testing and the design of the user experience Assist in troubleshooting and debugging issues related to user input and response handling

The user manual:

There are many extra features the chatbot provides for the user that creates a better experience for user such as:

1. **Spelling & Synonym Support:**
    - The chatbot accepts spelling mistakes and synonyms to make interactions smoother.
    - Example: Typing "plannet" instead of "planet" is still accepted!

    Quiz History:
    - Track your quiz results over time to see your improvement.
    - Example: "You scored 3 out of 5 in your last quiz. Try again?"
2. **Random Facts:**
    - Get interesting astronomy facts after each quiz.
    - Example: "Did you know? A day on Venus is longer than a year on Venus!"
3. **Color-Coding Questions:**
    - Correct answers turn green, incorrect answers turn red for instant feedback.

4. **Exit Quiz Anytime:**
    - Exit the quiz at any time and continue with the chatbot or close the server.
    - Example: "Exit quiz? Type 'continue' to resume or 'close' to stop."
5. **Webserver Integration:**
    - The chatbot allows users to interact with it through a web interface, making it more accessible and visually appealing.
    - Example: If a user types "Start Quiz," the backend communicates with the frontend and begins the quiz session seamlessly.

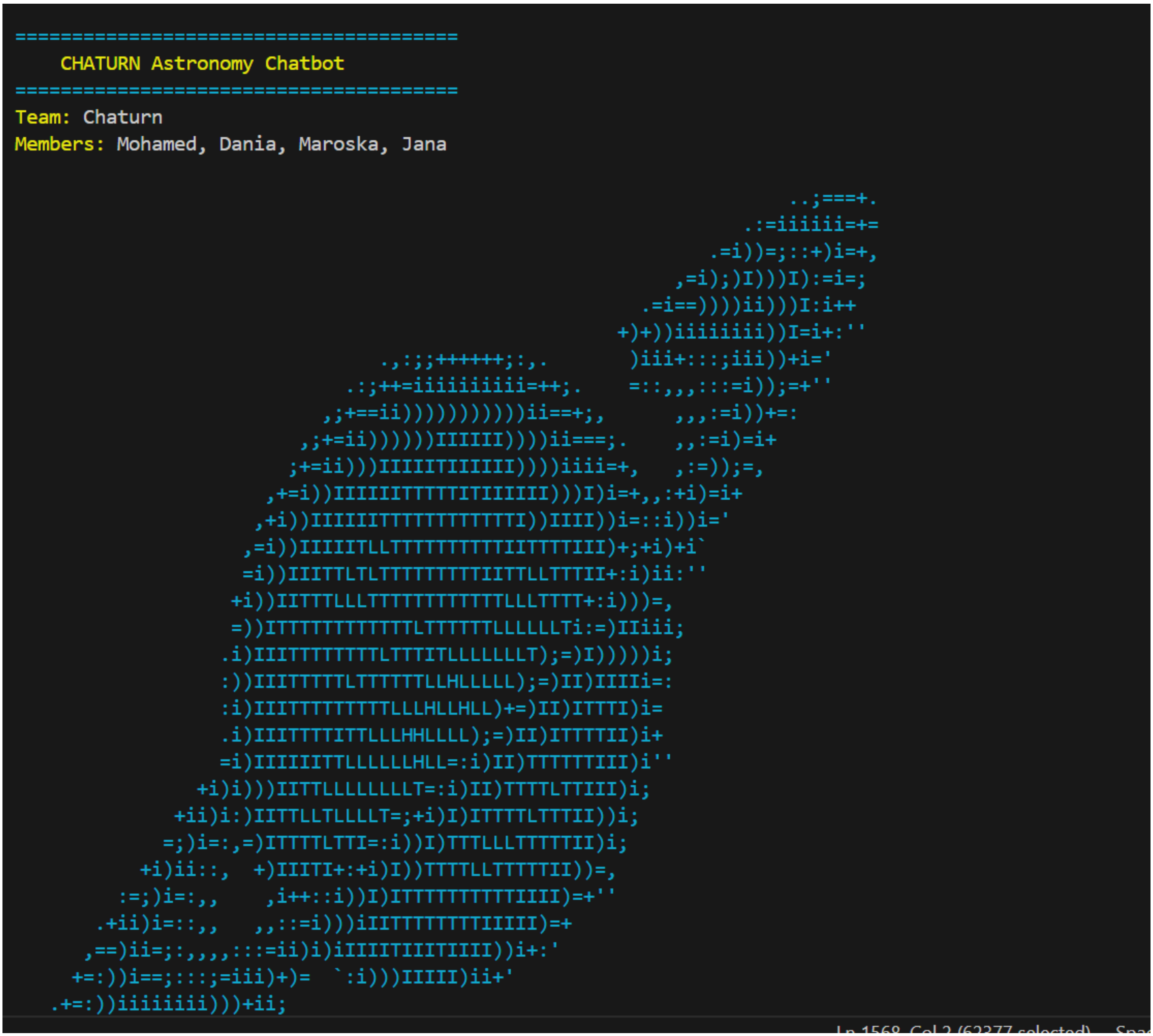
How to use our system?

1. **Launch the Chatbot**
    - Open the chatbot interface (through the web server or terminal).
    - The chatbot will greet you and prompt you to begin the interaction.
2. **Start a Conversation**
    - Type your first question or greeting (e.g., “Hello”, “Hi”, “What is astronomy?”).
    - The chatbot will respond with relevant information based on the greeting or query.
3. **Ask Astronomy-Related Questions**
    - You can ask questions about celestial bodies, space facts, astronomical events, and more.
    - Examples:
        - "What is a black hole?"
        - "Tell me about Mars."
        - "compare earth and vega”
4. **Take the Astronomy Quiz**
    - The chatbot will offer you the option to take a quiz about astronomy.
    - Answer the questions by typing your responses.
        - Correct answers are acknowledged with positive feedback and green colored.
        - Incorrect answers will prompt a brief explanation and red colored question.
5. **Enjoy Synonym and Spelling Correction**
    - The chatbot accepts common synonyms and spelling mistakes.
    - Example: If you type “planet” or “planit”, it will still recognize and respond appropriately.
6. **Use Special Commands**
    - **Exit Quiz:** You can exit the quiz at any time by typing “Exit” or “Quit”.
    - **Check Quiz History:** Type “Quiz History” to see a list of previously taken quizzes and your performance.
    - **Ask for Fun Facts:** Type “Tell me a fun fact” to get random astronomy-related facts.
7. **Customize the Experience**
    - **Personalized Questions:** Ask specific questions about your favorite planets, stars, or galaxies, and the chatbot will give tailored responses.
8. **Receive Feedback and Learn**
    - After each quiz, the chatbot will provide feedback on your performance.
    - Use this to learn and improve your knowledge of astronomy.
9. **Get Assistance**
    - If you encounter any issues, the chatbot will try to resolve them by offering suggestions or asking clarifying questions.
    - If you need a menu with the options just type help.
10. **End Session**
    - You can close the chatbot interface or exit the conversation at any time.
    - The chatbot will thank you for using the service and provide an option to continue or exit.

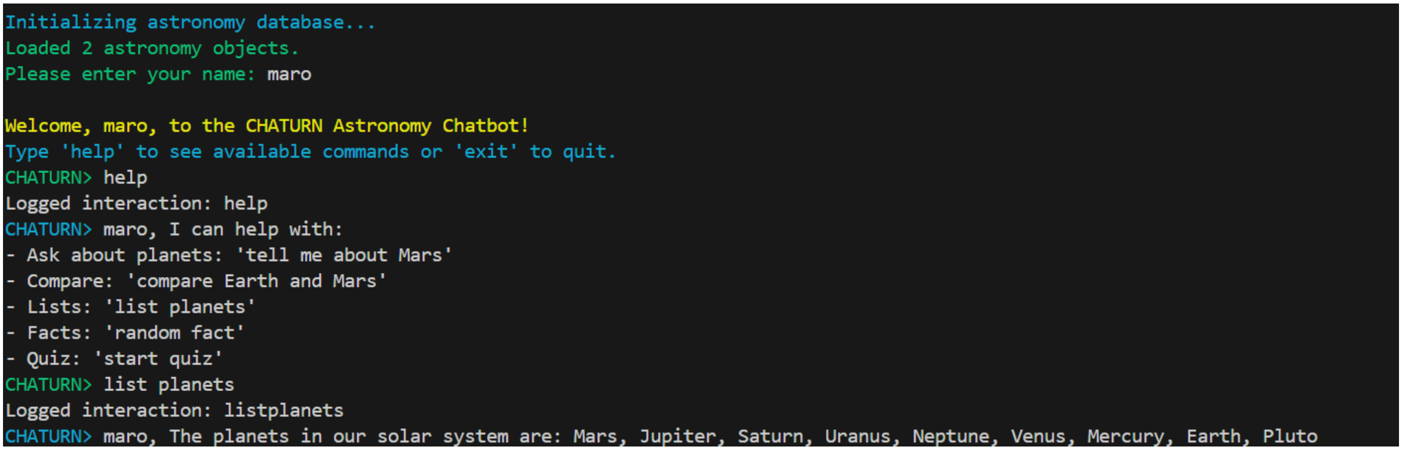
Additional Features:

- **Random Facts:** Type “Tell me a random fact” for interesting space facts.
- **Color-Coded Feedback:** Questions will be color-coded based on your answer (green for correct, red for incorrect).
- **Web Server Control:** You can close or keep the web server running while interacting with the chatbot

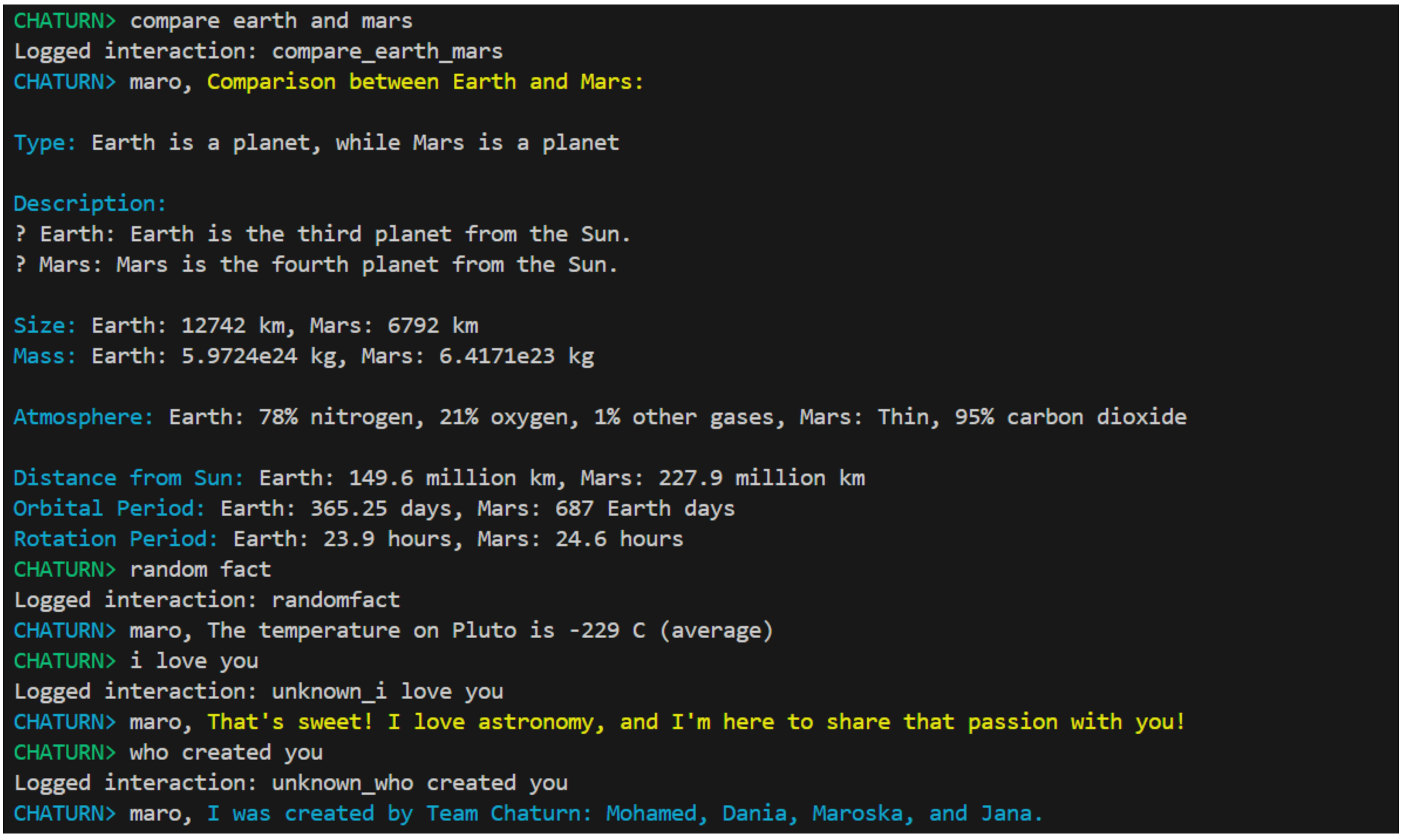
Testing and validation:



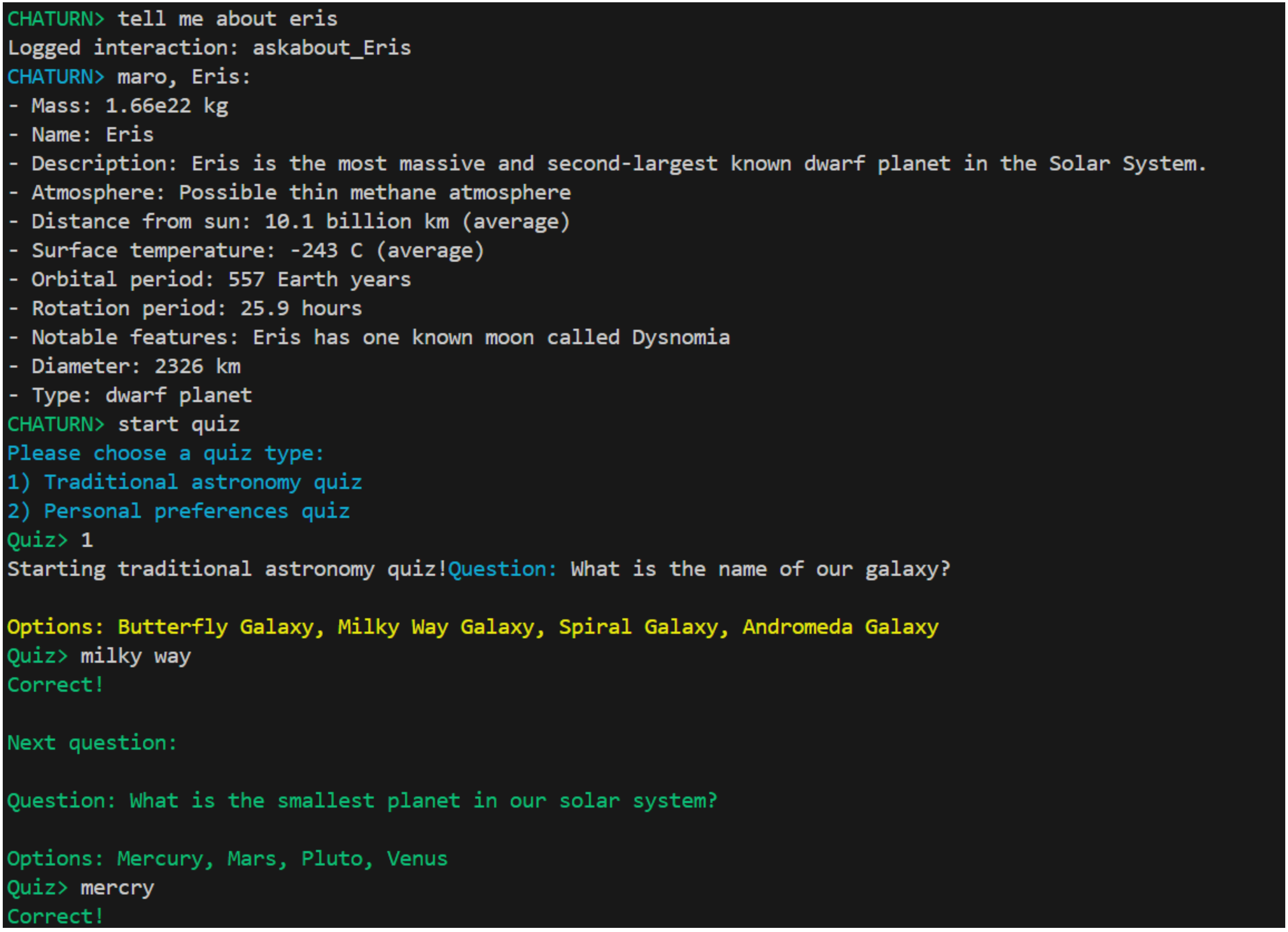
```
========================================
    CHATURN Astronomy Chatbot
========================================
Team: Chaturn
Members: Mohamed, Dania, Maroska, Jana

                                                                    ..;===+.
                                                                .:=iiiiii+=
                                                             .=i))=;::+)i=+,
                                                          ,=i);)I)))I):=i=;
                                                       .=i==))))ii)))I:i++
                                                     +)+))iiiiiiii))I=i+:''
                                .,:;;++++++;:,.       )iii+:::;iii))+i='
                             .:;++=iiiiiiiiii=++;.    =::,,,:::=i));=+''
                           ,;+==ii)))))))))))ii==+;,      ,,,:=i))+=:
                         ,;+=ii))))))IIIIII))))ii===;.    ,,:=i)=i+
                        ;+=ii)))IIIIITIIIIII))))iiii=+,   ,:=));=,
                      ,+=i))IIIIIITTTTTITIIIIIII)))I)i=+,,:+i)=i+
                     ,+i))IIIIIITTTTTTTTTTTTTI))IIII))i=::i))i='
                    ,=i))IIIIITLLTTTTTTTTTTTIITTTTTIII)+;+i)+i`
                    =i))IIITTLTLTTTTTTTTTTIITTLLTTTII+:i)ii:''
                   +i))IITTTLLLTTTTTTTTTTTTTLLLTTTT+:i)))=,
                   =))ITTTTTTTTTTTTLTTTTTTLLLLLLTi:=)IIiii;
                  .i)IIITTTTTTTTLTTTITLLLLLLLT);=)I)))))i;
                  :))IIITTTTTLTTTTTTLLHLLLLL);=)II)IIIIi=:
                  :i)IIITTTTTTTTTLLLHLLHLL)+=)II)ITTTI)i=
                  .i)IIITTTTITTLLLHHLLLL);=)II)ITTTTII)i+
                  =i)IIIIIITTLLLLLLHLL=:i)II)TTTTTTIII)i''
                +i)i)))IITTLLLLLLLLT=:i)II)TTTTLTTIII)i;
              +ii)i:)IITTLLTLLLLT=;+i)I)ITTTTLTTTII))i;
             =;)i=:,=)ITTTTLTTI=:i))I)TTTLLLTTTTTII)i;
           +i)ii::,  +)IIITI+:+i)I))TTTTLLTTTTTII))=,
         :=;)i=:,,    ,i++::i))I)ITTTTTTTTTTIIII)=+''
       .+ii)i=::,,   ,,::=i)))iIITTTTTTTTTIIIII)=+
      ,==)ii=;:,,,,,:::=ii)i)iIIIITIIITIIII))i+:'
     +=:))i==;:::;=iii)+)=  `:i)))IIIII)ii+'
   .+=:))iiiiiiii)))+ii;
```



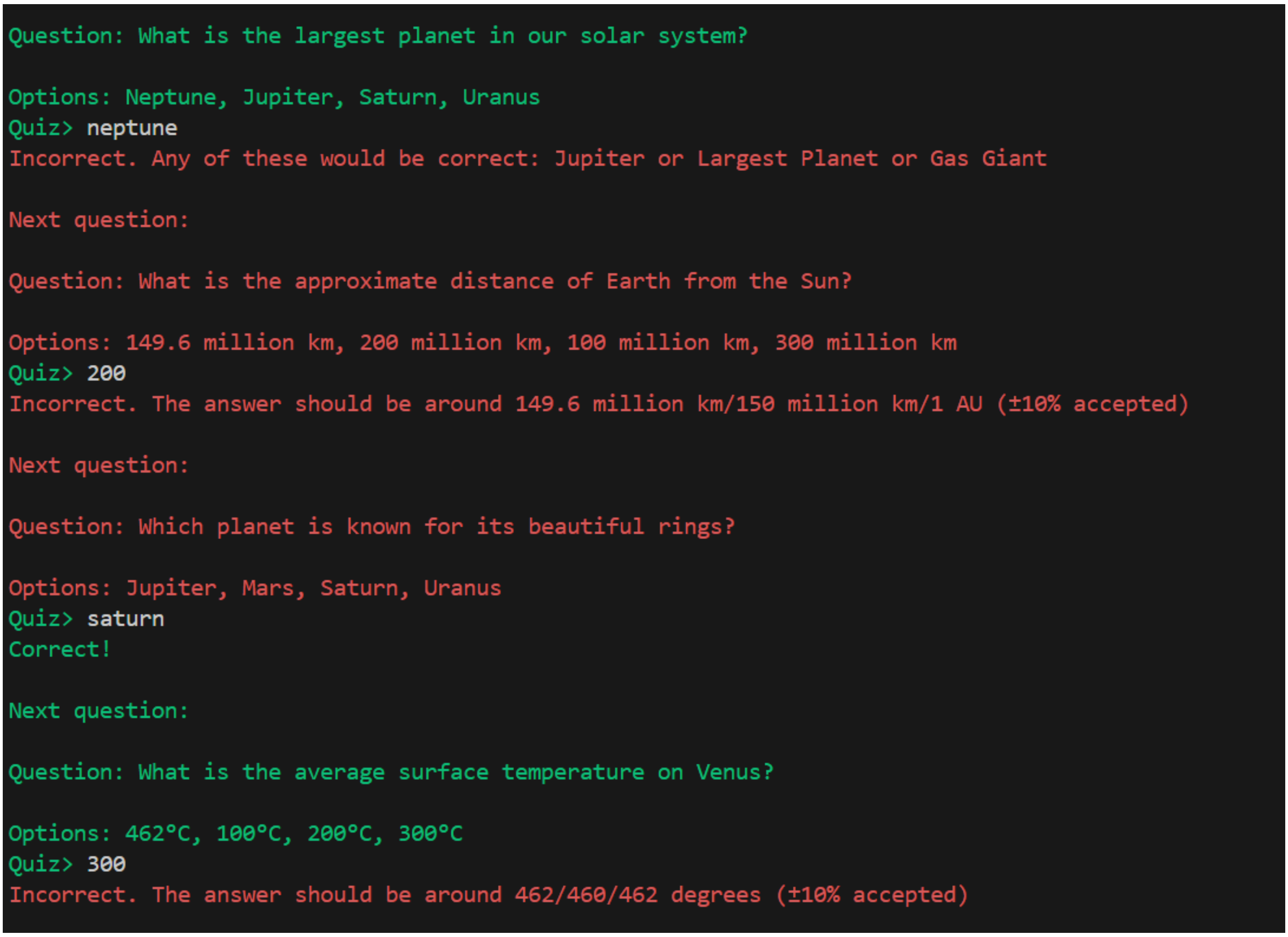
```
Initializing astronomy database...
Loaded 2 astronomy objects.
Please enter your name: maro

Welcome, maro, to the CHATURN Astronomy Chatbot!
Type 'help' to see available commands or 'exit' to quit.
CHATURN> help
Logged interaction: help
CHATURN> maro, I can help with:
- Ask about planets: 'tell me about Mars'
- Compare: 'compare Earth and Mars'
- Lists: 'list planets'
- Facts: 'random fact'
- Quiz: 'start quiz'
CHATURN> list planets
Logged interaction: listplanets
CHATURN> maro, The planets in our solar system are: Mars, Jupiter, Saturn, Uranus, Neptune, Venus, Mercury, Earth, Pluto
```



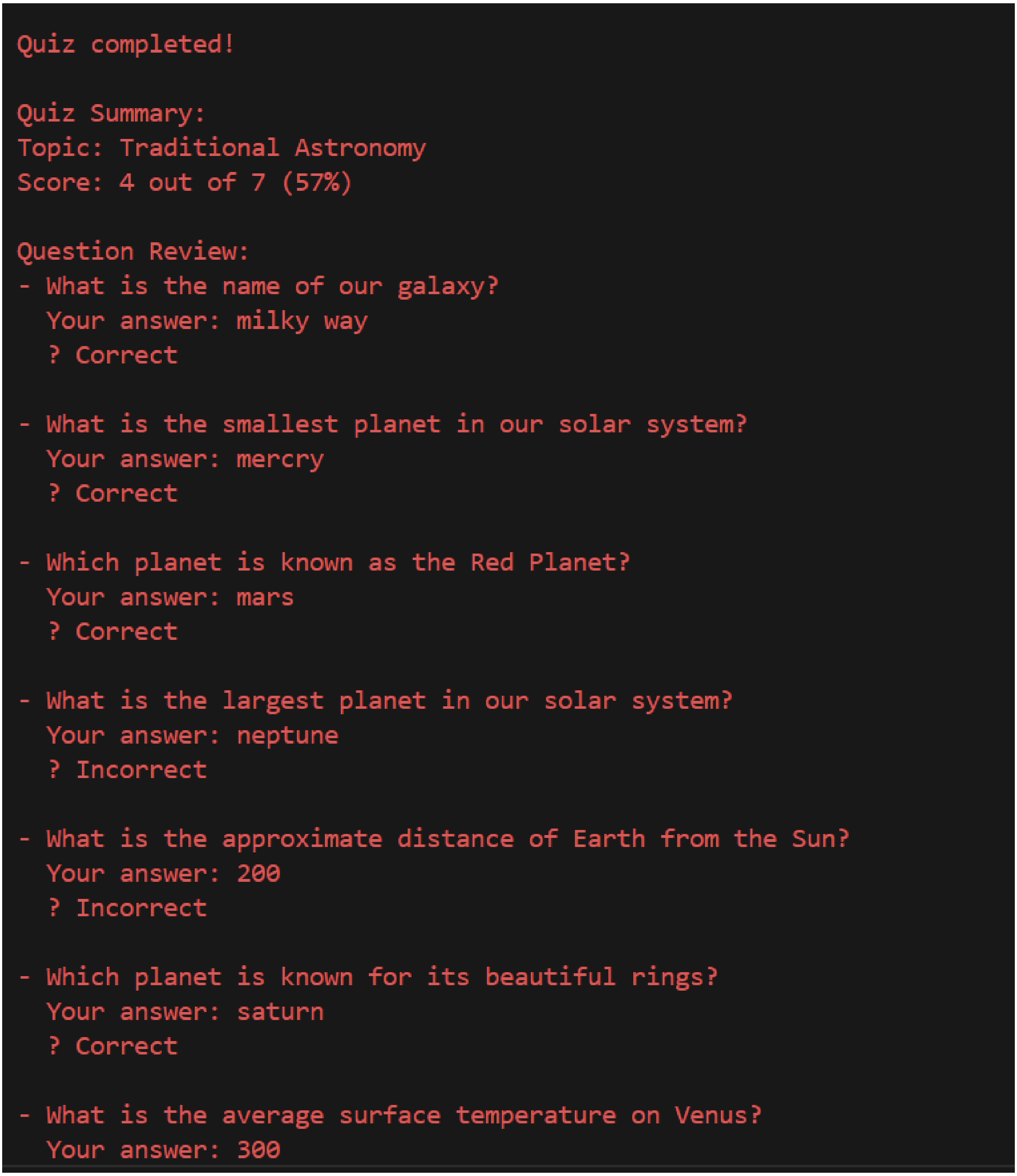
```
CHATURN> compare earth and mars
Logged interaction: compare_earth_mars
CHATURN> maro, Comparison between Earth and Mars:

Type: Earth is a planet, while Mars is a planet

Description:
? Earth: Earth is the third planet from the Sun.
? Mars: Mars is the fourth planet from the Sun.

Size: Earth: 12742 km, Mars: 6792 km
Mass: Earth: 5.9724e24 kg, Mars: 6.4171e23 kg

Atmosphere: Earth: 78% nitrogen, 21% oxygen, 1% other gases, Mars: Thin, 95% carbon dioxide

Distance from Sun: Earth: 149.6 million km, Mars: 227.9 million km
Orbital Period: Earth: 365.25 days, Mars: 687 Earth days
Rotation Period: Earth: 23.9 hours, Mars: 24.6 hours
CHATURN> random fact
Logged interaction: randomfact
CHATURN> maro, The temperature on Pluto is -229 C (average)
CHATURN> i love you
Logged interaction: unknown_i love you
CHATURN> maro, That's sweet! I love astronomy, and I'm here to share that passion with you!
CHATURN> who created you
Logged interaction: unknown_who created you
CHATURN> maro, I was created by Team Chaturn: Mohamed, Dania, Maroska, and Jana.
```



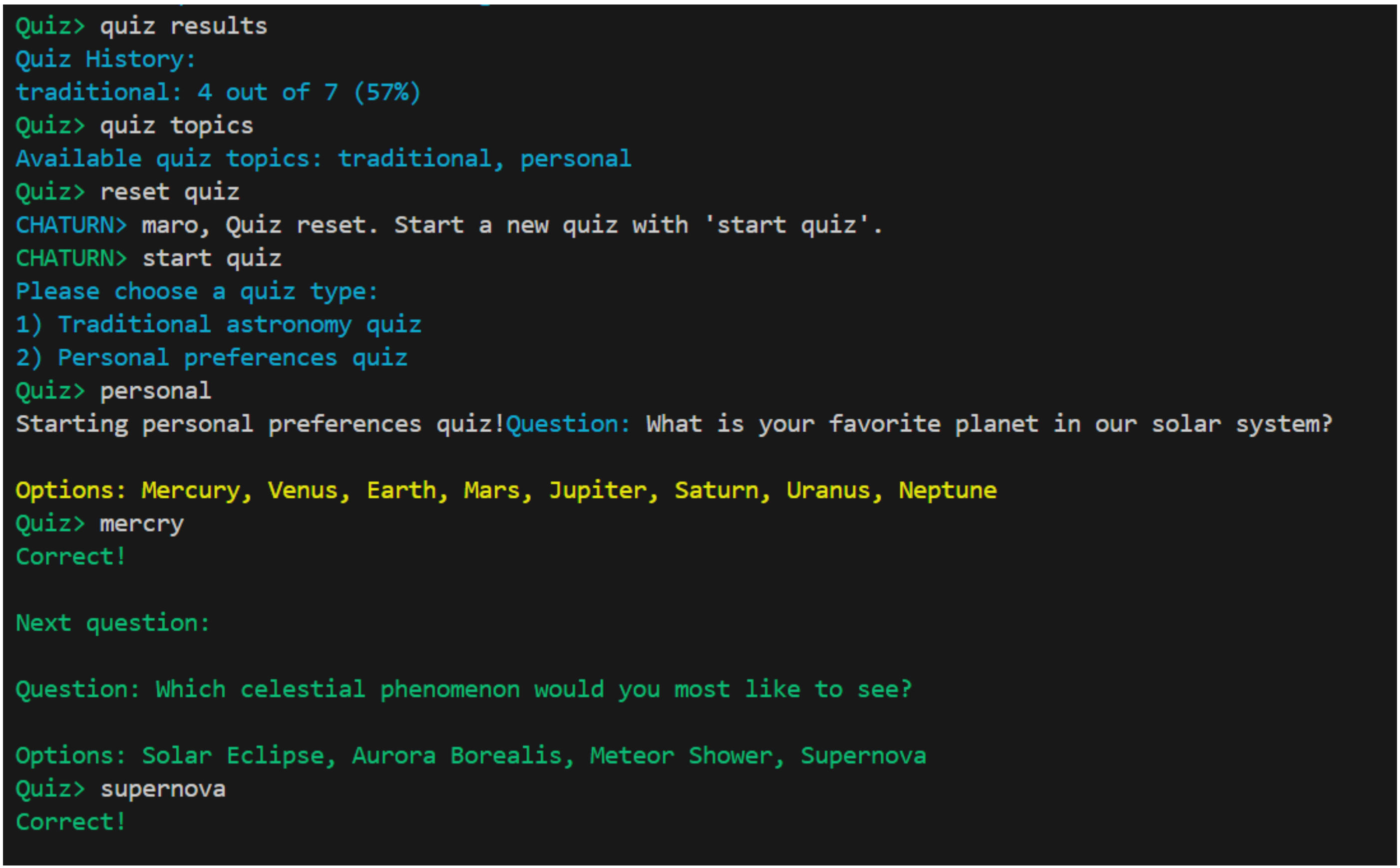
```
CHATURN> tell me about eris
Logged interaction: askabout_Eris
CHATURN> maro, Eris:
- Mass: 1.66e22 kg
- Name: Eris
- Description: Eris is the most massive and second-largest known dwarf planet in the Solar System.
- Atmosphere: Possible thin methane atmosphere
- Distance from sun: 10.1 billion km (average)
- Surface temperature: -243 C (average)
- Orbital period: 557 Earth years
- Rotation period: 25.9 hours
- Notable features: Eris has one known moon called Dysnomia
- Diameter: 2326 km
- Type: dwarf planet
CHATURN> start quiz
Please choose a quiz type:
1) Traditional astronomy quiz
2) Personal preferences quiz
Quiz> 1
Starting traditional astronomy quiz!Question: What is the name of our galaxy?

Options: Butterfly Galaxy, Milky Way Galaxy, Spiral Galaxy, Andromeda Galaxy
Quiz> milky way
Correct!

Next question:

Question: What is the smallest planet in our solar system?

Options: Mercury, Mars, Pluto, Venus
Quiz> mercry
Correct!
```



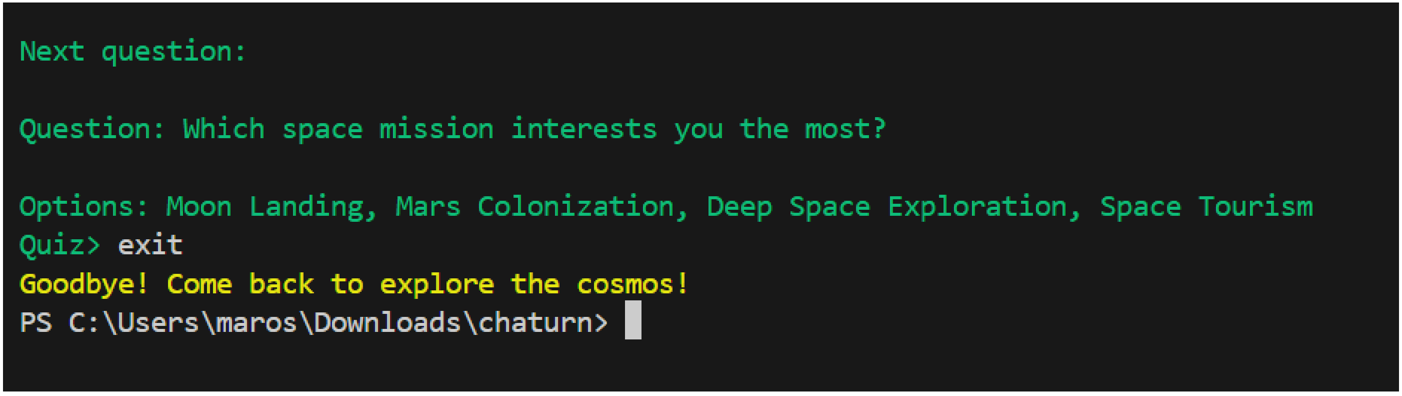
```
Question: What is the largest planet in our solar system?

Options: Neptune, Jupiter, Saturn, Uranus
Quiz> neptune
Incorrect. Any of these would be correct: Jupiter or Largest Planet or Gas Giant

Next question:

Question: What is the approximate distance of Earth from the Sun?

Options: 149.6 million km, 200 million km, 100 million km, 300 million km
Quiz> 200
Incorrect. The answer should be around 149.6 million km/150 million km/1 AU (±10% accepted)

Next question:

Question: Which planet is known for its beautiful rings?

Options: Jupiter, Mars, Saturn, Uranus
Quiz> saturn
Correct!

Next question:

Question: What is the average surface temperature on Venus?

Options: 462°C, 100°C, 200°C, 300°C
Quiz> 300
Incorrect. The answer should be around 462/460/462 degrees (±10% accepted)
```



```
Quiz completed!

Quiz Summary:
Topic: Traditional Astronomy
Score: 4 out of 7 (57%)

Question Review:
- What is the name of our galaxy?
  Your answer: milky way
  ? Correct

- What is the smallest planet in our solar system?
  Your answer: mercry
  ? Correct

- Which planet is known as the Red Planet?
  Your answer: mars
  ? Correct

- What is the largest planet in our solar system?
  Your answer: neptune
  ? Incorrect

- What is the approximate distance of Earth from the Sun?
  Your answer: 200
  ? Incorrect

- Which planet is known for its beautiful rings?
  Your answer: saturn
  ? Correct

- What is the average surface temperature on Venus?
  Your answer: 300
```



```
Quiz> quiz results
Quiz History:
traditional: 4 out of 7 (57%)
Quiz> quiz topics
Available quiz topics: traditional, personal
Quiz> reset quiz
CHATURN> maro, Quiz reset. Start a new quiz with 'start quiz'.
CHATURN> start quiz
Please choose a quiz type:
1) Traditional astronomy quiz
2) Personal preferences quiz
Quiz> personal
Starting personal preferences quiz!Question: What is your favorite planet in our solar system?

Options: Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune
Quiz> mercry
Correct!

Next question:

Question: Which celestial phenomenon would you most like to see?

Options: Solar Eclipse, Aurora Borealis, Meteor Shower, Supernova
Quiz> supernova
Correct!
```



```
Next question:

Question: Which space mission interests you the most?

Options: Moon Landing, Mars Colonization, Deep Space Exploration, Space Tourism
Quiz> exit
Goodbye! Come back to explore the cosmos!
PS C:\Users\maros\Downloads\chaturn>
```

The conclusion and challenges:

Complex Input Parsing: Handling misspellings, synonyms, and ambiguous inputs.

Synonym and Spelling Correction: Ensuring variations in wording were processed correctly.

Quiz Logic and State Management: Implementing interactive quiz functionality and tracking progress.

Web Server Integration: Synchronizing backend and frontend for smooth communication.

Error Handling: Preventing unexpected errors and managing edge cases.

Testing and Debugging: Identifying and resolving bugs in the system.

The overall astronomy chatbot project delivers an engaging and user-friendly platform for learning about space. With features like quiz functionality, synonym and spelling handling, and color-coded feedback, it creates an interactive and personalized experience. Additional functions, such as random facts and quiz history, enhance user engagement. Overall, the chatbot is a valuable educational tool, offering both flexibility and insight, making space exploration accessible and enjoyable for everyone.