MOONSHOT REPORT

Paul MARIS - 21/08/23

PRESENTATION OF THE PROJECT

INTRODUCTION TO MOON

Introducing Moon, a new programming language designed with simplicity and accessibility at its core. Unlike many existing languages that can be complex and intimidating for beginners, Moon strips away unnecessary complexity and focuses on intuitive, minimalist syntax. By using familiar symbols and natural language constructs, it brings the power of coding to those who may have never thought it possible.

PURPOSE AND DESIGN GOALS

The primary purpose of this language is to make programming an enjoyable and comprehensible experience, especially for children and coding novices. The design goals are as follows:

1. Simplicity: Offer a gentle learning curve by using straightforward and minimalist syntax.
2. Intuitiveness: Create a language that resonates with natural language understanding, reducing the cognitive load for new learners.
3. Flexibility: Allow for growth and exploration by including essential programming concepts like variables, functions, loops, and conditions.
4. Engagement: Include features like human-readable explanations of code execution and potential gamification elements to keep learners engaged.

TARGET AUDIENCE

Moon is aimed at children, educators, and anyone new to programming who wants to understand and experiment with coding in an

unthreatening environment. Here's how different groups might use the language:

- Children: As a first introduction to programming, allowing them to create simple programs and understand foundational concepts.
- Educators: As a teaching tool in schools to introduce programming concepts in a more engaging and comprehensible way.
- Novice Programmers: For self-learners or those in coding bootcamps who need a gentle introduction to programming before moving on to more complex languages.
- Creative Exploration: For artists, writers, and other creatives who want to experiment with coding without getting bogged down in syntax.
- Advanced Programmers: As a tool for rapid prototyping and experimentation, allowing them to quickly test out ideas and concepts.

In essence, this language is more than just a programming language; it's a bridge to the world of computational thinking and a gateway to the vast possibilities of programming.

ANALYSIS AND SPECIFICATIONS

The creation of the Moon programming language required meticulous planning and implementation. This section outlines the key aspects that were essential in the development of the language.

LEXICAL STRUCTURE

The lexical structure of Moon defines the fundamental building blocks of the language, known as tokens. These include keywords, identifiers, literals, operators, and various symbols. The lexical structure is governed by specific rules that determine how these elements are recognised, separated, and interpreted by the compiler.

A robust lexer was implemented to process the source code and break it down into these tokens, allowing for efficient parsing and interpretation.

SYNTAX

Syntax refers to the set of rules that dictate how tokens are combined to form valid statements and expressions within Moon. This includes the rules for constructing variables, functions, control structures, and more.

The syntax of Moon was designed to be intuitive and beginner-friendly, providing clear guidelines on how to structure the code. A parser was created to analyse the tokens and generate an Abstract Syntax Tree (AST), which represents the hierarchical structure of the code.

SEMANTICS

Semantics deals with the meaning of the constructs in the language. It defines how the syntactically correct statements are to be executed or evaluated.

In Moon, the semantics encompasses the behavior of various constructs such as data types, control flow, and function calls. Careful consideration was given to ensure that the semantics align with the goals of the language, providing a consistent and logical execution model.

If you are interested in learning more about the syntax, semantics, and detailed workings of Moon, you are welcome to explore the comprehensive documentation available on GitHub.

SOFTWARE ARCHITECTURE CHOICES

WHY USING PYTHON AND PLY LIB

The choice of Python as the implementation language for Moon was driven by its simplicity, readability, cross-platform capabilities, rapid development speed, and extensive library support.
The PLY library, a well-known standard for over 20 years, was selected for its compatibility with Python, flexibility in defining tokens and grammar rules. Together, Python and PLY provide a solid foundation that balances ease of use with powerful functionality, making them a logical choice for language parsing and design.

EXECUTION MODEL

The execution model of Moon is based on a three-step process: Lexical Analysis, Syntax Analysis, and Interpretation. This model, a standard approach utilised by most programming languages, was chosen for its efficiency and effectiveness in translating the high-level language into machine code. The methodology aligns with industry best practices, ensuring that the language design is both robust and maintainable. By adhering to this widely accepted standard, the development process is streamlined, and the resulting language is more likely to be compatible with existing tools and technologies.

TYPE SYSTEM

Moon's type system is designed with simplicity and intuition in mind. It includes features like Type Inference and Type Coercion. This system was chosen to provide a seamless and user-friendly coding experience.

ERROR HANDLING

Error handling in Moon is designed to be transparent and informative. Clear and human-readable error messages are provided, and the program stops execution upon encountering an error. This approach was chosen to assist developers in quickly identifying and resolving issues.

EXPLANATION OF ALGORITHMS USED

The algorithms used in the Moon programming language include parsing and interpretation. Parsing is done using a combination of lexical analysis and syntax analysis, converting the source code into an abstract syntax tree (AST). Interpretation involves traversing the AST and executing the corresponding code.

LEXER

The lexer, or lexical analyser, is responsible for scanning the source code and converting it into a stream of tokens. These tokens represent the smallest meaningful units in the language, such as keywords, identifiers, literals, and operators. In Moon, the lexer recognises the specific syntax and structure of the language, including indentation and special characters. It's an essential step in translating the high-level source code

into a format that can be further analysed and interpreted. The lexer's efficiency and accuracy are paramount, as errors at this stage can lead to subsequent issues in parsing and interpretation.

PARSING

Parsing is a critical part of any programming language, and in Moon, it's done using a combination of lexical analysis and syntax analysis. Lexical analysis breaks the source code into tokens, as handled by the lexer. Syntax analysis then organises these tokens into a structured AST, based on the grammatical rules of the language. This hierarchical tree represents the grammatical structure of the code and serves as a bridge between the lexical analysis and interpretation. Parsing errors are handled gracefully, providing informative error messages to assist the developer.

INTERPRETATION

Interpretation involves traversing the AST and executing the corresponding code. This approach allows for a more flexible and dynamic execution of the code, making it easier to implement features like error handling and debugging. Unlike compilation, which translates the entire program into machine code before execution, interpretation executes the code directly from the AST. This allows for a more interactive and iterative development process, as changes to the code can be tested immediately without the need for recompilation.

RELEVANCE TO THE PROBLEMS SOLVED

Moon's commitment to accessibility and ease of use positions it as a compelling solution for those taking their first steps into programming. Whether for educational purposes or personal exploration, Moon provides a nurturing environment that fosters creativity, curiosity, and confidence in the realm of coding.

TESTS AND PRODUCTION LAUNCH

TESTING METHODOLOGIES

Testing is a crucial part of the development process, ensuring that the language functions as intended. In Moon, testing is done using unit tests and integration tests. More about how it works, on github.

LAUNCH STRATEGY

The production launch of Moon involves a phased rollout, starting with a beta available for everyone on github. This approach allows for thorough testing and feedback, ensuring that any issues are addressed before a full public release.

DEVELOPMENTS MADE AND FUTURE ENHANCEMENT

CURRENT FEATURES

Moon currently supports a range of features, including variables, functions, loops, conditions, and more. These features are designed to provide a comprehensive and accessible programming experience.

More information available on github.

FUTURE ENHANCEMENTS

Future enhancements for Moon include additional data types, improved error handling, development and enhancement of the standard library and potential integration with educational platforms. These enhancements aim to make Moon even more versatile and appealing to a broader audience.

ANALYSIS OF PROJECT MANAGEMENT

PROJECT MANAGEMENT METHODOLOGIES

The development of Moon follows agile project management methodologies, allowing for flexibility and adaptability in the development process. Regular sprints and iterative development ensure that the project stays on track and aligns with the overall goals and vision.

Moon's development cycle is divided into regular sprints, each focusing on specific features, enhancements, or bug fixes. These sprints are guided by a prioritised backlog, which is constantly updated to reflect the project's current needs. Iterative development ensures that the project

stays on track and aligns with the overall goals and vision. It also allows for continuous testing and integration, ensuring the quality and reliability of the codebase.

Moon's development leverages GitHub for version control, issue tracking, and project management. The GitHub project provides a centralised platform for collaboration, code review, and documentation. It also facilitates transparent communication. Scheduled management is handled through GitHub's project boards, allowing for clear visualisation of tasks, priorities, and progress.

Moon's development places a strong emphasis on the user experience, aiming to create a language that is both accessible and engaging. The design of the language has been tested with 12-year-old kids to ensure that the design and the syntax is intuitive. This unique approach to user testing has helped shape a programming language that is not only suitable for experienced developers but also inviting for younger audiences and those new to coding.

GLOSSARY

- AST (Abstract Syntax Tree): A hierarchical tree-like structure that represents the grammatical structure of the source code. It serves as an intermediate step between lexical analysis and interpretation.
- Indentation: The use of whitespace to define the structure of the code. In Moon, indentation is used to signify code blocks and control flow.
- Lexer: Also known as a lexical analyser, this component of the compiler scans the source code and breaks it down into tokens, the smallest meaningful units in the language.
- Lexical Analysis: The process of converting the source code into a series of tokens. This is done by the lexer.
- Parser: A component that analyses tokens and organises them into an AST, based on the grammatical rules of the language.
- Parsing: The process of analysing the tokens and generating an AST, which represents the hierarchical structure of the code.
- PLY Library: A parsing library used in conjunction with Python to define tokens and grammar rules.
- Semantics: The meaning of the constructs within the language, defining how syntactically correct statements are to be executed or evaluated.

- Syntax: The set of rules that dictate how tokens are combined to form valid statements and expressions within the language.
- Tokens: The fundamental building blocks of the language, including keywords, identifiers, literals, operators, and other symbols.
- Type Coercion: The automatic or implicit conversion of a value from one type to another.
- Type Inference: The automatic deduction of the data type of an expression within the programming language.

FURTHER DOCUMENTATION

Feel free to delve into the extensive documentation available on this project's GitHub repository, where you'll find detailed information, insights, and explanations.