YARTBML Language Architecture

This document outlines the architecture of the YARTBML language, detailing its interpreter components.

Lexer

The lexer is responsible for the initial processing of the source code. It performs the following tasks:

- Token definition: Identifying keywords, delimiters, operators, and literals.
- Rule definition: Establishing regular expressions that define tokens.
- Input processing: Reading the input string incrementally, typically in chunks or individual words.
- Tokenization: Generating tokenized output based on the input string.
- Error handling: Defining conditions for handling errors when input does not match defined rules.

Parser

The parser translates the tokenized output from the lexer into an Abstract Syntax Tree (AST) representing the structure of the program. Key functionalities include:

- AST creation: Constructing a hierarchical representation of the program structure.
- Integration with lexer: Utilizing lexer output as input for parsing.
- Token analysis: Reading and analyzing the current token and peeking at the next token in the sequence.
- Parsing rules: Defining rules for parsing tokens and their relationships within the program structure. For example, specifying that the ‘fn’ keyword must be followed by a left parenthesis and then a block statement.
- Error handling: Managing errors that occur when parsing tokens based on defined rules.

Evaluation

The evaluation phase interprets the AST generated by the parser to execute the program’s logic. This phase involves traversing the AST and performing actions corresponding to the nodes encountered. Key tasks during evaluation include:

- Interpretation: Executing the program logic represented by the AST nodes.
- Data manipulation: Handling variables, expressions, and control flow statements to produce desired outcomes.

- Execution environment: Managing the runtime environment, such as memory allocation, function invocation, and error handling.
- Result generation: Producing output or side effects based on the program's execution.

REPL (Read-Eval-Print Loop)

The REPL provides an interactive environment for developers to experiment with YARTBML code. It consists of the following components:

- Read: Accepting user input, typically YARTBML expressions or statements.
- Eval: Evaluating the input code, executing it and producing results.
- Print: Displaying the results of the evaluation to the user.
- Loop: Repeating the process, allowing users to enter and evaluate additional code sequentially.

The REPL facilitates rapid prototyping, debugging, and exploration of YARTBML code without the need for compiling or running entire programs.