Project Report: Simple Arithmetic Expression Evaluator with Syntax Tree Visualization

Introduction

This project involves the creation of a simple arithmetic expression evaluator and visualizer using Python and the Tkinter library. The program reads arithmetic expressions, parses them into a syntax tree, evaluates the result, and displays the syntax tree graphically.

Components

1. **Tokenization (Lexer):**

- The Lexer class breaks down an input string into tokens representing integers, arithmetic operators (+, -, *, /), parentheses, and the end of the file (EOF).
- It handles whitespace and identifies errors when encountering invalid characters.

2. **Parsing (Parser):**

- The Parser class constructs an Abstract Syntax Tree (AST) from the tokens provided by the Lexer.
- It supports the four basic arithmetic operations and respects operator precedence and parentheses through recursive parsing methods (**factor**, **term**, **expr**).

3. **Syntax Tree Nodes:**

- The project defines two types of nodes: **BinOp** for binary operations (with left and right children and an operator) and **Num** for numeric values.
- These nodes are used to build the AST.

4. **Evaluation:**

- The **visit** function recursively evaluates the AST by traversing it and performing the operations represented by each node.
- It handles basic arithmetic operations and raises an error for division by zero.

5. **User Interface:**

- The program uses Tkinter to create a GUI that displays the syntax tree and the result of the evaluated expression.
- The tree is displayed in a textual, indented format to represent the hierarchy of operations

Functionality

- **Lexer:** Converts input strings into tokens, identifying numbers and operators while skipping whitespace.
- **Parser:** Constructs an AST that represents the input expression's structure.
- **AST Nodes:** Represent the elements of the parsed expression.
- **Evaluation:** Calculates the result by traversing the AST.
- **Tkinter GUI:** Displays the syntax tree and the result.

Usage

To use the program, run it in a Python environment. It continuously prompts the user for arithmetic expressions, evaluates them, and displays the syntax tree and result. For instance:

```
calc > (3+2) * 4 – 6/3
```

This input would be parsed, and evaluated, and the result along with the syntax tree would be displayed in a GUI window.

Conclusion

This project demonstrates a basic interpreter for arithmetic expressions, including lexical analysis, parsing, and evaluation. The use of Tkinter provides a visual representation of the syntax tree, aiding in understanding the structure of expressions and the order of operations. The implementation is modular and can be extended to support more complex expressions or additional features in the future.