**2020040**

**2020397**

**Milestone 1 Report: Understanding the Grammar and Implementing Input Handling**

**Overview**

This report covers the development of foundational components for an arithmetic expression parser as outlined in Milestone 1. The main objectives are to document the grammar rules of arithmetic expressions and to implement a basic input handling system that tokenizes user input according to these rules. This groundwork is crucial for the robustness and functionality of the subsequent parsing stages.

**1. Grammar Rules Documentation**

The arithmetic expressions handled by our parser adhere to a simple yet strict grammar involving digits, operators, and parentheses. Here’s a detailed breakdown:

* **Digits**: Any decimal number, including integers (e.g., 2, 13) and floating-point numbers (e.g., 3.14). These form the basic operands for arithmetic operations.
* **Operators**: Four basic arithmetic operators are considered:
  + Addition (+)
  + Subtraction (-)
  + Multiplication (\*)
  + Division (/)
* **Parentheses**: Used to explicitly define the precedence of operations (e.g., (3 + 2) \* 5).

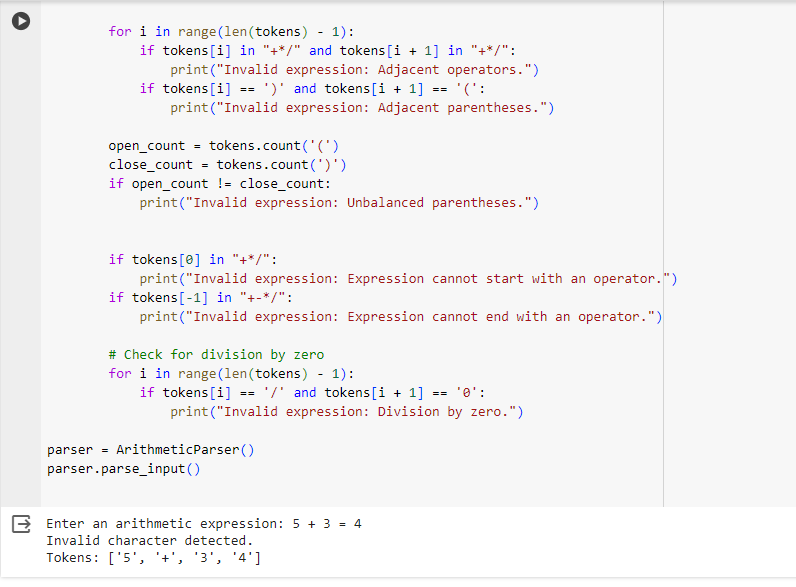
These elements combine to form valid arithmetic expressions such as 4 + 18 / (9 - 3).

**2. Input Handling Implementation**

The implementation involves the following steps, executed within a Python class named ArithmeticParser:

* **Tokenization**: We utilize regular expressions to extract tokens representing numbers, operators, and parentheses from the user’s input.
* **Validation**: Each extracted token is validated against a set of allowed characters and structures to ensure it adheres to the expected grammar of arithmetic expressions.
* **User Input**: Through a command-line interface, the user is prompted to enter an arithmetic expression, which is then processed to tokenize and validate.

**Code**

****

**Error handling (Validate tokens)  
  
Testing and Validation**

The ArithmeticParser was tested with various input scenarios to ensure it accurately tokenizes and validates expressions. Tests included correct inputs like 12 \* 3.5 and erroneous inputs such as 12 $ 3.5, where the latter correctly triggered a validation error.

**Conclusion**

The successful completion of Milestone 1 establishes a robust foundation for our arithmetic expression parser. The grammar is well-documented and understood, and the initial input handling system is capable of processing and validating user inputs efficiently. This setup prepares us for the more complex tasks of parsing these tokens into a structured format that can be evaluated mathematically in subsequent milestones.