**Stage 1:Basic Calculator**

This project is a simple mathematical expression evaluator built in Python. It takes arithmetic expressions as input, processes them using a parser, and evaluates them step by step. It supports basic operations, parentheses, and operator precedence, making it a lightweight yet powerful calculator.

**Features**

* Basic arithmetic operations: +, -, \*, /
* Supports parentheses for grouping expressions
* Handles unary minus (-x) correctly
* Respects operator precedence (multiplication and division before addition and subtraction)
* Converts expressions from infix to postfix notation using the Shunting Yard Algorithm
* Evaluates postfix expressions using a stack-based approach
* Supports exponentiation (^) and modulus (%)
* Displays error messages for invalid inputs
* Accepts user input interactively via the terminal

**How It Works**

Tokenizing the Input

The program first breaks down the input into numbers, operators, and parentheses using regular expressions. This step ensures the program correctly understands what the user entered.

Converting to Postfix (Reverse Polish Notation)

Instead of evaluating the expression directly, we convert it from infix (standard mathematical notation) to postfix notation using the Shunting Yard Algorithm. This makes it easier for the program to process operations in the correct order.

Evaluating the Postfix Expression

Once we have the postfix expression, we use a stack to compute the result. Numbers are pushed onto the stack, and when an operator is encountered, the relevant numbers are popped, the operation is performed, and the result is pushed back onto the stack.

**Run**

To run the program, open a terminal and execute:

*/python3 implementation.py*

Then, enter mathematical expressions like these:

e.g Enter a mathematical expression (or 'exit' to quit): (3 + 4) \* 2

Result: 14.0

Handling Negative Numbers:

e.g Enter a mathematical expression (or 'exit' to quit): -5 + 3

Result: -2.0

e.g Enter a mathematical expression (or 'exit' to quit): - -3

Result: 3.0

Power and Modulus:

e.g Enter a mathematical expression (or 'exit' to quit): 2 ^ 3

Result: 8.0

e.g Enter a mathematical expression (or 'exit' to quit): 10 % 3

Result: 1.0

Error Handling

If the user enters an invalid expression, the program will return an error message.

e.g Invalid characters: abc + 5 → Error: Invalid character

e.g Mismatched parentheses: (5 + 2 → Error: Malformed expression

e.g Division by zero: 5 / 0 → Error: Division by zero

**Stage 2:Boolean Logic**

This stage builds upon the basic arithmetic calculator developed in Stage 1 by adding support for Boolean expressions and strict type checking. Users can now input logical comparisons and Boolean operations such as *true and false*, *!true*, and *5 < 10*.

The interpreter evaluates these expressions while ensuring strict type consistency.i.e., numbers and Booleans cannot be mixed in operations.

**Features**

* Boolean values: true, false (case-insensitive)
* Logical operators: *! (NOT) , and, or*
* Comparison operators: *==, !=*  between same types (number-number or bool-bool). *<, >, <=, >=* between numbers only.
* Strict type checking: Prevents mixing of numbers and Booleans
* Informative error messages when types don't match
* Interactive CLI input

**How it Works**

Parsing Boolean Expressions

The tokenizer recognizes Boolean literals (true, false) and logical operators (and, or, !). These tokens are integrated seamlessly into the Shunting Yard algorithm and postfix evaluator used in Stage 1.

Type-Safe Evaluation

Each operation is validated:

* Arithmetic ops (+, -, etc.) are allowed only for numbers
* Logical ops (and, or, !) are allowed only for Booleans
* Comparisons (==, <, etc.) are allowed only between same types

If an invalid operation is detected (e.g., true + 5), the interpreter will raise a TypeError with a clear explanation.

**Run**

To run the program, open a terminal and execute:

*/python3 implementation.py*

as we did in stage 1.

Enter Boolean expressions like:

e.g Enter a mathematical expression (or 'exit' to quit): true and !false

Result: True

e.g Enter a mathematical expression (or 'exit' to quit): 5 < 10

Result: True

e.g Enter a mathematical expression (or 'exit' to quit): !(5 > 2 == false)

Result: False

Valid Examples Index:

***Boolean Expression Result***

|  |  |
| --- | --- |
| True == false | False |
| True != false | true |
| !true | false |
| True and false | False |
| False or true | true |
| (5 > 2) and (2 < 4) | true |
| 7 == 7 | True |
| !(false or false) | true |

Invalid Examples Index

**Boolean Expr. Result**

|  |  |
| --- | --- |
| true + 1 | Error: '+' cannot operate on BOOLEAN and NUMBER |
| 5 < false | Error: '<' cannot compare NUMBER and BOOLEAN |
| true == 1 | Error: '==' cannot compare BOOLEAN and NUMBER |
| false \* 3 | Error: '\*' cannot operate on BOOLEAN and NUMBER |
| true and 1 | Error: 'and' requires two BOOLEANS |

1. The interpreter uses strict typing, meaning types are not coerced (e.g., true ≠ 1).
2. Booleans and numbers are handled by the same tokenizer and expression parser, but their evaluation is strictly separated.
3. This design ensures predictable behavior, avoids bugs, and aligns with best practices for language design.

# **Installation and Setup**

To run this interpreter, follow these steps:  
1. Clone the repository:  
 git clone *https://github.com/Loukas89/Parser-Expression\_Interpreter.git*  
2. Navigate to the project directory:  
 cd *Parser-Expression\_Interpreter*   
 3.Run the interpreter using:  
 */python3 implementation.py*

**Interactive Mode**

The interpreter supports an interactive mode, prompting users to input

expressions directly via the terminal.