최종 보고서

🧾 SimpleCalc Final Project Report

1. Project Overview

Project Name: SimpleCalc

Version: 1.0

Developer: Jo Hojin (Department of Computer Science)

Date: October 2025

Purpose

SimpleCalc is a lightweight console-based calculator program that allows users to perform basic arithmetic, percentage (%), and modulus (%%) operations easily.

The project was designed and implemented following software engineering principles — from requirement analysis and UML modeling to implementation and testing.

---

2. Requirements Summary

2.1 Functional Requirements (FR)

ID	Requirement	Description	Priority
FR-1	Basic Arithmetic	Supports addition, subtraction, multiplication, division	★★★★★
FR-2	Percentage Calculation	Calculates percentage values using `%` postfix (e.g., `200*12%`)	★★★★☆
FR-3	Modulus Operation	Performs remainder calculations using `%%` (e.g., `7 %% 3`)	★★★★☆
FR-4	Calculation History	Stores up to 5 recent results in memory	★★★☆☆

ID	Requirement	Description	Priority
FR-5	Error Handling	Displays friendly error messages without terminating program	★★★★☆
FR-6	CLI Interface	User-friendly console interaction with commands (`help`, `history`, `exit`)	★★★★★

2.2 Non-Functional Requirements (NFR)

ID	Category	Description
NFR-1	Performance	Calculation results should appear within 0.5 seconds
NFR-2	Reliability	Program must remain stable during invalid inputs
NFR-3	Usability	Interface must be clear and beginner-friendly
NFR-4	Portability	Compatible with Python 3.12+ on macOS and Windows
NFR-5	Security	Prevent arbitrary code execution; allow only arithmetic expressions

3. System Design

3.1 Architecture Overview

The program follows a **modular, object-oriented architecture**:

```
CLI → Calculator → Parser → Evaluator → Formatter
               ↘ HistoryManager → HistoryItem
```

- **CLI:** Handles user interaction and command input/output
- **Calculator:** Core controller that coordinates parsing, evaluation, and formatting
- **Parser:** Converts input expressions into safe, evaluable forms
- **Evaluator:** Interprets and computes the parsed expressions using Python's AST
- **Formatter:** Applies rounding and output formatting rules

- **HistoryManager:** Stores and retrieves up to 5 calculation records

3.2 UML Diagrams

Class Diagram

(As designed — showing Calculator aggregation with Parser, Evaluator, Formatter, and association with HistoryManager)

Sequence Diagram

- **UC-1 Calculation:**
  `CLI → Calculator → Parser → Evaluator → Formatter → HistoryManager → CLI`
- **UC-2 History Retrieval:**
  `CLI → Calculator → HistoryManager → CLI`

4. Implementation

4.1 Programming Environment

Item	Description
Language	Python 3.12
Editor	Visual Studio Code / Terminal
Version Control	Git + GitHub
Testing Framework	pytest

4.2 Key Features

- **Percentage & Modulus:**
  - `Parser` automatically converts `12%` → `(12*0.01)` and `%%` → `%`
- **Decimal Accuracy:**
  - `Formatter` rounds results to 2–4 decimal places using `Decimal`
- **Error Recovery:**
  - Invalid input triggers `Error: 잘못된 입력입니다.` but program continues running
- **Interactive CLI:**

  - Command-based interface with color formatting, help menu, and history table

5. Testing & Validation

5.1 Test Cases

All major components were unit tested using `pytest`.

Module	Test File	Key Cases	Result
Parser	`test_parser.py`	% / %% / invalid chars	✅ Passed
Evaluator	`test_evaluator.py`	+ - * / %	✅ Passed
Formatter	`test_formatter.py`	rounding rules	✅ Passed
History	`test_history.py`	add/list/max=5	✅ Passed
Calculator	`test_calculator.py`	full pipeline	✅ Passed

5.2 Sample CLI Test

```
› 5 * 3
= 15.00
› 200 * 12%
= 24.00
› 7 %% 3
= 1.00
› history
 1  7 %% 3     1.00
 2  200*12%    24.00
 3  5*3        15.00
```

All test cases passed (13/13), confirming system stability and correctness.

6. Version Control Summary

Commit	Description
1st	Initial project setup & Git config
2nd	Implemented Parser/Evaluator/Formatter/History
3rd	Added CLI with help/history/clear commands

Commit	Description
4th	Added pytest test cases
5th	Updated README.md & final polishing

7. Conclusion

The SimpleCalc project successfully achieved all functional and non-functional requirements.

Through structured requirement analysis, UML modeling, modular implementation, and automated testing,

the program demonstrates reliability, clarity, and maintainability — serving as a practical reference for small-scale software engineering workflows.

8. Future Improvements

- Add parentheses-based expression precedence display
- Save/load history from file
- Expand to GUI version (Tkinter / PyQt)
- Support scientific functions (√, sin, log, etc.)

9. Attachments

- Source Code: https://github.com/sumb-10/SimpleCalc-python-