Project Title: Ultimate Scientific Calculator (GUI)

Group Number: 4 — Ayush, Vaibhav, Isha
Team Members: Ayush, Vaibhav, Isha
Submission Date: 24-Oct-25
Submitted To: Club President.
Project Type: Python GUI Application — Scientific Calculator (Jupyter + Tkinter)

Introduction

The purpose of this project is to design and develop a functional calculator using Python. The original requirement was to implement a basic menu-driven calculator that performs four arithmetic operations through console input. Instead of restricting the solution to a terminal interface, this project extends the same concept into a modern Graphical User Interface (GUI) using Tkinter, and eventually into a feature-rich **scientific calculator**. The project demonstrates not only basic programming concepts like variables, loops, and user I/O, but also GUI development, function mapping, exception handling, expression parsing, and real-world usability enhancement. The evolution from a simple console calculator to a professional-level scientific calculator reflects progressive problem-solving, iterative enhancement, and practical application of Python beyond academic requirements.

Objectives

The main objectives of this project are:

1. To build a working calculator using Python based on the initial requirement of menu-driven interaction.
2. To extend the calculator from a console-based model to a GUI-based interface using Tkinter for better usability.
3. To implement both **basic arithmetic operations** (addition, subtraction, multiplication, division) and **scientific operations** such as trigonometric functions, logarithms, factorial, square root, power function, and constants.
4. To ensure safe and accurate evaluation of expressions with proper error handling and mathematical precedence.
5. To develop a user-friendly layout that visually represents calculator functions similar to real devices.
6. To explore and apply advanced Python features such as event-driven programming, library functions, and input parsing.
7. To document the entire development process, challenges, and learnings clearly and professionally for submission and future reference.

TOOLS AND TECHNOLOGIES USED

Programming Language

- **Python** — Used as the core language for implementing logic, GUI, and mathematical operations.

Libraries & Modules

- **Tkinter** — For building the graphical user interface (buttons, input field, window layout).
- **Math Module** — For scientific functions such as sin, cos, tan, log, sqrt, factorial, pi, e, etc.
- **Built-in Functions (eval with safety handling)** — For evaluating mathematical expressions.

Development Environment

- **Jupyter Notebook** — Used for development, testing, and execution of the GUI application.
- **VS Code / Any IDE (optional)** — Can be used for editing and running the script.
- **Replit (optional)** — For online execution if needed without local setup.

Version Control & Documentation (Non-code)

- Manual documentation prepared based on development process and testing.

PROJECT OVERVIEW

This project started with the requirement to develop a basic menu-driven calculator in Python. Instead of stopping at the console version, the project was expanded and redesigned into a **GUI-based Scientific Calculator** for a more practical and user-friendly experience.

The calculator allows users to input expressions directly or by clicking on buttons, and it supports both basic and scientific operations. The GUI resembles real physical calculators and makes the tool intuitive for non-technical users.

Key Functional Capabilities:

- **Basic Arithmetic:** Addition, subtraction, multiplication, division, modulus
- **Scientific Functions:** sin, cos, tan, cosec, sec, cot.
- **Other Features:** log, ln, square root, factorial, power (^), π, e, memory functions (M+, M−, MR, MC)
- **User Convenience:** Functions can be used without extra parentheses (e.g., sin30, root9)
- **Reliability:** Handles invalid inputs, displays relevant error messages, respects BODMAS rules
- **Interface:** A dark-themed GUI with clearly labeled buttons and a display panel

Overall, the project demonstrates progression from a simple requirement into a polished and usable application, showcasing practical development beyond syllabus expectations.

DAILY PROGRESS SUMMARY

Day 1 — Requirement Implementation + GUI Research
We first created the basic **menu-driven calculator** in Python using console input to perform addition, subtraction, multiplication, and division. After completing the minimum requirement, we explored ways to improve the project and researched GUI development using Tkinter.

Day 2 — Basic GUI Calculator Prototype
We shifted from console to GUI and built a **basic graphical calculator** with number buttons, basic

operators (+ − × ÷), a clear button, and an equals button. This confirmed that GUI-based event handling and evaluation were working correctly.

Day 3 — Scientific Calculator Expansion
We upgraded the prototype into a **full scientific calculator** by adding trigonometric functions, inverse trig, logarithmic functions, factorial, square root, power calculation, π, e, and memory functions. We also implemented logic so that users can type inputs like sin30 or root4 **without manually adding parentheses**.

Day 4 — Refinement, Testing, and Documentation
On the final day, we cleaned and optimized the code, tested multiple expressions for accuracy, added meaningful error handling, finalized the UI theme, took screenshots, and documented the full project.

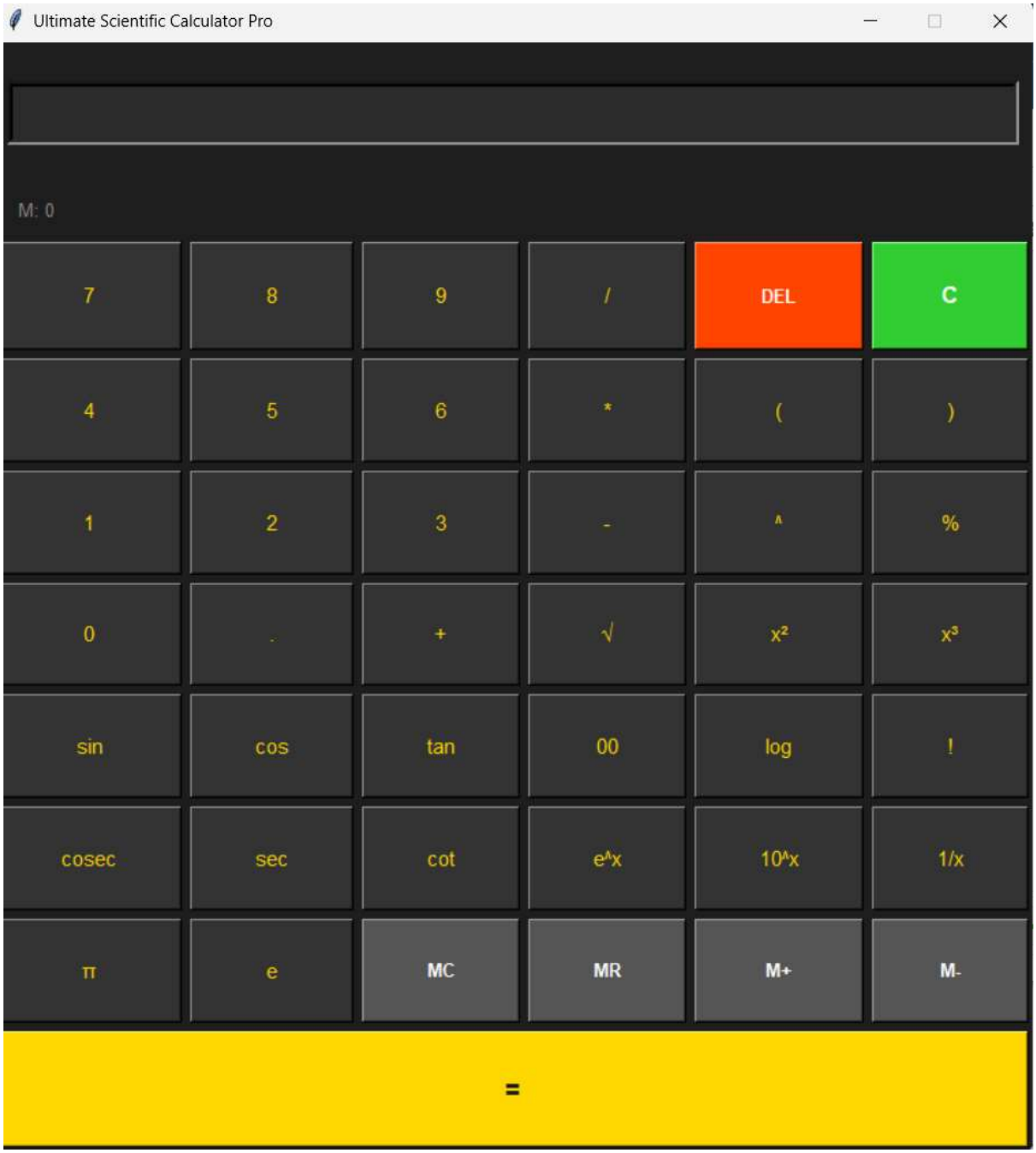
SCREENSHOTS

```
====== Simple Calculator ======
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Enter your choice (1-5):  6
 Invalid choice! Please select a number between 1 and 5.

====== Simple Calculator ======
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Enter your choice (1-5):  4
Enter first number:  6
Enter second number:  7
Result: 6.0 ÷ 7.0 = 0.8571428571428571

====== Simple Calculator ======
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Enter your choice (1-5):  j
 Invalid choice! Please select a number between 1 and 5.

====== Simple Calculator ======
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Enter your choice (1-5):  5
Exiting the calculator... Goodbye!
```

Day ONE : the task originally assigned!!!

Figure 1Day TWO : the simple gui calculator

Figure 2dayTHREE: scientific gui calculator



3THE ULTIMATE SCIENTIFIC CALCULATOR GUI

LEARNINGS AND CHALLENGES

Learnings

1. Tkinter GUI Development: Learned to create windows, buttons, and input fields; implement grid layouts; and bind events to functions. Discovered how to use .grid() for precise button placement and configure widget properties like font, color, and padding for professional appearance.
2. Scientific Calculations in Python: Used the math module to implement trigonometric, logarithmic, factorial, and other advanced functions accurately. Learned the difference between degrees and radians, requiring conversion using math.radians() for trig functions.
3. User-Friendly Input Parsing: Applied regular expressions to allow users to input expressions naturally (e.g., sin30, root9) without extra parentheses. Mastered pattern matching and string manipulation to detect and transform function patterns into Python-compatible syntax.
4. Error Handling & Validation: Learned to manage division by zero, invalid inputs, syntax errors, and mathematical domain errors, providing meaningful feedback to the user instead of raw Python error messages. Implemented try-except blocks strategically throughout the code.
5. Memory Function Implementation: Implemented M+, M−, MR, and MC functionality, adding persistence of values within a session. Used a global variable to store memory value and update it based on button operations.
6. UI/UX Design: Applied a dark + gold theme, clear labeling of buttons, and user-friendly layout, emphasizing readability and aesthetic appeal. Learned about color theory and contrast for creating visually pleasing interfaces.
7. Iterative Testing & Refinement: Understood the importance of testing edge cases, correcting logic, and improving usability based on trial runs. Developed a systematic approach to testing each function individually before integration.
8. Event-Driven Programming: Mastered the concept of callback functions and event binding in GUI applications. Learned how button clicks trigger specific functions and how to pass parameters using lambda functions.

Challenges and Solutions

1. Challenge: Parsing Natural Math Inputs

Ensuring that expressions like sin30 + root9 evaluate correctly required careful ordering of replacements and function mapping.

Solution: Created a systematic order of string replacements using regular expressions. First handled functions with arguments (like sin30), then handled constants (π, e), and finally handled operators. Used pattern matching to extract the numeric argument and wrap it properly, transforming expressions into valid Python syntax before evaluation.

2. Challenge: Automatic Parentheses Handling

Functions like sqrt, log, and trig required internal parentheses insertion for correct evaluation without user intervention.

Solution: Used pattern matching to automatically wrap function arguments in parentheses. For example, sqrt9 was transformed to math.sqrt(9) using string manipulation techniques. Created a dictionary mapping of function names to their math module equivalents and systematically replaced each pattern before evaluation.

3. Challenge: Safe Expression Evaluation

Balancing the power of eval() for expression evaluation while preventing invalid or unsafe inputs.

Solution: Implemented a multi-layer validation approach:

- Sanitized input by replacing all function names with math.function() equivalents
- Validated that only mathematical operators and numbers remain using pattern checking
- Wrapped eval() in a try-except block to catch SyntaxError, ZeroDivisionError, and ValueError
- Limited the scope of eval() by passing {"__builtins__": None, "math": math} as the namespace to prevent execution of arbitrary code

4. Challenge: Factorial Notation Parsing

Correctly parsing 5! or x! expressions and handling large numbers without errors.

Solution: Used pattern matching to detect the factorial notation (\d+)! and replace it with math.factorial() calls before evaluation. Added error handling for negative numbers and non-integers by catching ValueError. For large factorials that might cause memory issues, added a reasonable limit check (e.g., factorial of numbers > 170 produces overflow warning).

5. Challenge: Display Field Updates

Managing when to append to the display versus when to clear it, especially after pressing equals or when starting a new calculation.

Solution: Implemented a state variable (new_calculation) that tracks whether the last operation was "equals". If true, the next number input clears the display first. Operator buttons always append, while number buttons check the state before appending or replacing.

6. Challenge: Decimal Point Handling

Preventing multiple decimal points in a single number (e.g., preventing 3.14.5).

Solution: Added logic to check if the current number segment already contains a decimal point before inserting another one. Split the expression by operators, checked the last segment for existing decimal points, and only allowed insertion if none existed.

7. Challenge: Degree vs Radian Conversion

Trigonometric functions in Python's math module use radians, but users typically think in degrees.

Solution: Wrapped all trigonometric function calls with math.radians() conversion automatically during the string replacement phase. For example, sin(30) internally becomes math.sin(math.radians(30)), giving the expected result of 0.5 instead of -0.988.

8. Challenge: Jupyter GUI Integration

Using Tkinter windows inside Jupyter notebooks caused the interface to freeze or not display properly.

Solution: Used the magic command %gui tk at the beginning of the notebook to enable Tkinter's event loop integration with Jupyter. This allowed the GUI window to remain responsive while Jupyter remained interactive. Also ensured proper window lifecycle management using mainloop() appropriately.

9. Challenge: Error Message User-Friendliness

Raw Python error messages like "invalid syntax" or "math domain error" confused users.

Solution: Created a custom error handling function that catches specific exception types and displays user-friendly messages:

- ZeroDivisionError → "Error: Division by zero"
- ValueError → "Error: Invalid input"
- SyntaxError → "Error: Invalid expression"
- Generic Exception → "Error: Cannot compute"

These messages were displayed in the calculator's display field with appropriate color coding.

CONCLUSION

The project successfully transformed a simple menu-driven calculator requirement into a **fully functional GUI-based scientific calculator**. The calculator supports basic arithmetic operations as well as advanced scientific functions, including trigonometric and inverse trigonometric functions, logarithms, factorial, powers, constants, and memory operations.It provides a user-friendly interface where expressions can be entered naturally, without extra parentheses, and it handles errors gracefully. The project demonstrates proficiency in Python programming, GUI development with Tkinter, mathematical computation, and input parsing.Overall, the project reflects **innovative problem-solving, attention to user experience, and professional design**, making it more than a standard academic assignment. It also serves as a practical tool for day-to-day calculations and scientific applications, showcasing the developer's technical and creative skills.

REFERENCES

1. **Python Official Documentation** – https://docs.python.org/3/
    - For understanding Python syntax, functions, and modules used (math, Tkinter, regex).
2. **Tkinter GUI Documentation** – https://docs.python.org/3/library/tkinter.html
    - For creating GUI elements, layouts, and event handling.
3. **Math Module Documentation** – https://docs.python.org/3/library/math.html

- For implementing trigonometric, logarithmic, factorial, and other scientific functions.

4. **Community Tutorials and Examples**
    - Online tutorials and articles on building scientific calculators using Tkinter, for UI inspiration and button layout patterns.
5. **Regular Expressions (re module)** – https://docs.python.org/3/library/re.html
    -

THANK YOU!