VIT Bhopal University

Bhopal-Indore Highway

Kothrikalan, Sehore, Madhya Pradesh - 466114

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ACADEMIC YEAR: 2025-2026

:VITyarthi Project Report:

Scientific Calculator GUI using Python and Tkinter

FALL SEMESTER (2025 – 2026)

Submitted By

Name :- **Yash Raj**

Reg No :- **25BAI11046**

Course Code :- **CSE1021**

Course Title :- **Introduction to Problem Solving and Programming**

Index

1.Introduction

2. Problem Statement

3. Functional Requirements

4. Non-functional Requirements

5. System Architecture

6. Design Diagrams

- ○ Use Case Diagram
- ○ Workflow Diagram
- ○ Sequence Diagram
- ○ Class/Component Diagram
- ○ ER Diagram (if storage used)

7. Design Decisions & Rationale

8. Implementation Details

9. Screenshots / Results

10. Testing Approach

11. Challenges Faced

12. Learnings & Key Takeaways

13. Future Enhancements

14. References

:INTRODUCTION:

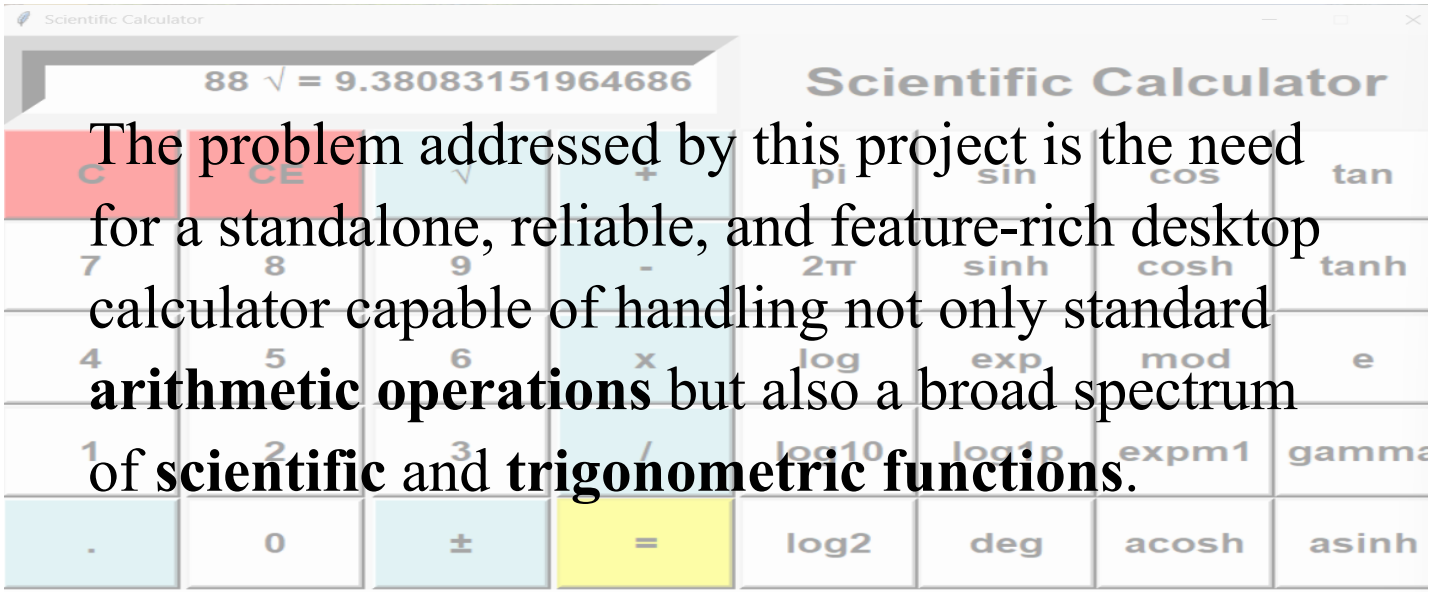
The objective of this project is to apply core programming concepts, specifically

- Object-Oriented Programming **(OOP)**
- Graphical User Interface **(GUI)** development, to create a functional and comprehensive **Scientific Calculator.**

The application is built using **Python** with its **standard GUI library**, **Tkinter**, integrated with the **standard math module** for advanced functions. This project serves as a demonstration of applying technical skills to design, implement, and validate a user-friendly software tool.

:PROBLEM STATEMENT:

Modern computing environments require tools that can perform rapid, complex calculations without relying on external web services.



The problem addressed by this project is the need for a standalone, reliable, and feature-rich desktop calculator capable of handling not only standard **arithmetic operations** but also a broad spectrum of **scientific** and **trigonometric functions**.

The solution must prioritize clear input/output structure and a logical workflow.

:FUNCTIONAL REQUIREMNTS:

The project design ensures the inclusion of at least **three major functional modules** , fulfilling the requirement for **Simulation or Visualization** and **CRUD Operations** (for data processing).

- **Module 1: Basic Arithmetic Engine (CRUD Operations)**

This module handles fundamental calculations and state management.

Operations: Supports Addition (+), Subtraction (-), Multiplication (x), Division (/), and Modulo (mod).

Input Control: Includes Clear Entry (C), All Clear (CE), Decimal Point (.), and Sign Change (±).

Result Calculation: The sum_of_total() function executes the pending operation (self.valid_function()) and resets the state for the next sequence.

Module 2: Scientific Functions (Simulation/Visualization)

This module integrates Python's math library to perform complex operations on the currently displayed number.

- **Trigonometric:** sin, cos, tan (and their hyperbolic counterpart's sin, cosh, tanh), operating on degrees (converted to radians internally).
- **Logarithmic/Exponential:** log, log2, log10, log1p, exp, expm1.
- **Special Functions/Constants:** Square root (√), Gamma function (gamma), constants (pi, tau, e), and unit conversion (degrees).

Module 3: User Interface & I/O Control

This module manages the display and button mapping.

- **Clear I/O Structure:** A single large **Entry** widget (txtDisplay) is used to show the current number, the live expression, and the final result.
- **Button Mapping:** All buttons are mapped to specific methods within the Calc class, providing a direct, event-driven user interaction model.

:Non-functional Requirements:

The project adheres to at least four (4) specified non-functional requirements.

Requirement	Description	Fulfillment in Project
Usability	The GUI must be intuitive, responsive, and easy for a novice user to navigate.	Standardized two-panel calculator layout (basic vs. scientific functions), large, high-contrast buttons, and live expression display.
Performance	Basic and scientific calculations must execute instantaneously.	Built on native Python and Tkinter, and relies on the optimized math module, ensuring near-instantaneous computation speeds.

Error Handling Strategy	The system must gracefully handle calculation errors without crashing the application.	The sum_of_total() and valid function() methods include try-except blocks to catch ZeroDivisionError and general exceptions, displaying user-friendly error messages like "Error: Division by zero."
Logging or Monitoring	The calculator should be able to capture or log the sequence of operations performed.	The self. Expression variable tracks the live mathematical expression. (This is a foundational component for the **Future Enhancement** of full Screen Recording/Session Capture).

: System Architecture :

The project employs a simple, single-tier, **Client-Side Architecture** based on the Model-View-Controller (MVC) pattern adapted for Tkinter.

- **View (GUI)**: The main script defines the visual elements (root, calc Frame, txtDisplay, and all Button widgets).
- **Controller/Model (Logic)**: The Calc class encapsulates the application state (total, current, op) and business logic (operation, sum_of_total, pi, sin, etc.).

This architecture is effective as it is a standalone, computation-heavy application that does not require external data storage.

```
from tkinter import *
import math

root = Tk()
root.title("Scientific Calculator")
root.configure(background='white')
root.resizable(width=False, height=False)
root.geometry("944x568+200+50")  # width of the calculator

calc = Frame(root)
calc.grid()
```

:Design Diagrams:

Use Case Diagram

Use Case: Perform Calculation

- Actor: User
- Use Cases: Enter Number, Select Basic Operation, Select Scientific Function, Calculate Result, Clear Input.
- Relationships: All use cases relate to the "Perform Calculation" primary use case.

Workflow Diagram

Diagram: Shows the state transitions of the calculator.

- Start: App opens, txtDisplay = "0".
- Flow: [Enter Number]⟶[Select Operator]⟶ [Enter Number] ⟶ [Click "="] ⟶ [Display Result, Reset State] OR [Select Scientific Function] [Display Result] ⟶ [Reset State].
- Error Path: [Attempt Division by Zero] ⟶ [Display "Error"].

Class/Component Diagram

Components:

1. main (Component): Handles Tkinter initialization, global button binding, and instantiation of the Calc class.
2. Calc (Class):
    - Attributes: total, current, op, expression, check_sum, input_value, result.
    - Methods (Simplified): __init__, numberEnter, operation, valid_function, sum_of_total, Clear_Entry, All_Clear_Entry, calculate_and_update, and individual function handlers (sin, log, etc.).

Sequence Diagram

Sequence: Perform Simple Addition (5 + 3 = 8)

1. User: Clicks '5'.
2. Calc: numberEnter(5) ⟶ updates current="5".
3. User: Clicks '+'.
4. Calc: operation('+') ⟶ sets total=5.0, op='+', check_sum=True.

5. User: Clicks '3'.
6. Calc: numberEnter(3) ⟶ updates current="3".
7. User: Clicks '='.
8. Calc: sum_of_total() ⟶ calls valid_function().
9. Calc: valid_function() ⟶ checks op='+' $\rightarrow$ total = 8.0.
10. Calc: sum_of_total() ⟶ display("8.0").

:Design Decisions & Rationale:

Decision	Rationale
Python & Tkinter	Tkinter is the native, standard Python GUI library, ideal for creating simple, self-contained desktop applications without external dependencies. This choice supports rapid prototyping and simplicity.
Object-Oriented Design (The Calc Class)	Encapsulating all calculator logic (state variables and methods) within the Calc class ensures modularity and clean state management. This prevents global variable pollution and allows for proper resetting of calculation status.

Combined Display (txtDisplay)	Using one input field for both input and expression tracking simplifies the UI and aligns with how modern physical calculators operate.
calculate_and_update Helper Method	Refactored the scientific functions into a single helper method to avoid redundancy, centralize error checking, and ensure consistent state updates for calculation chaining.

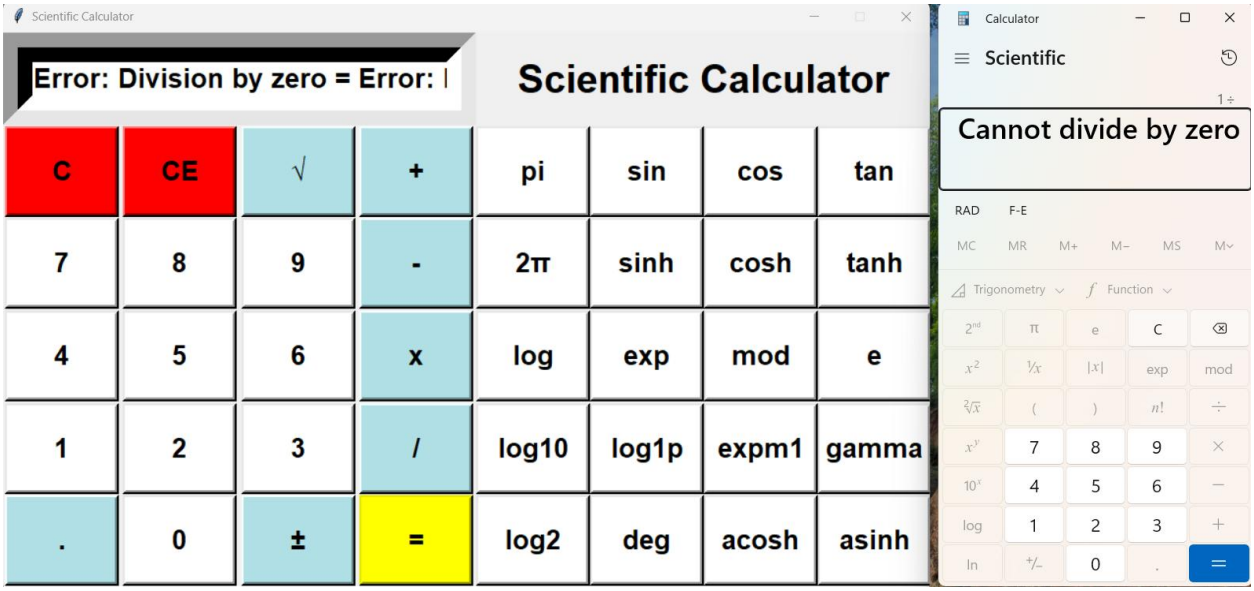
:Implementation Details:

The implementation follows a modular and clean structure. The script is organized into three main sections:

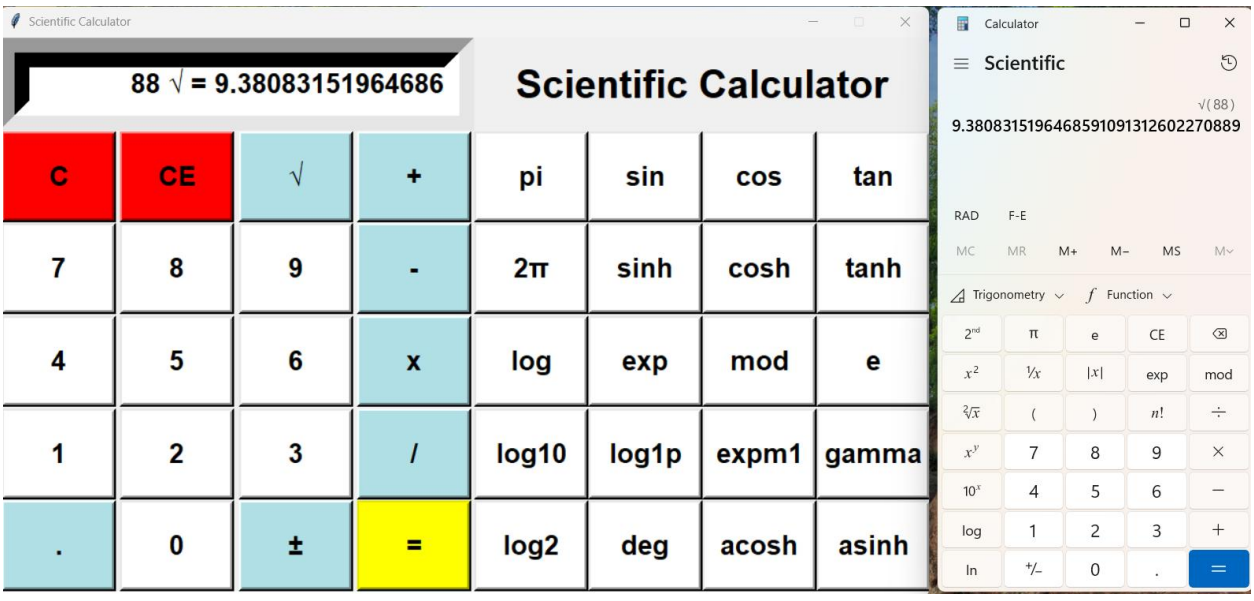
1. **Initialization**: Sets up the main Tkinter window (root), frame (calc), and the display entry field (txtDisplay).
2. Calc **Class**: The core logic container.
    - The use of self.total (the accumulator) and self.current (the current input number) is key to handling sequential operations.
    - The boolean flags (self.input_value, self.check_sum, self.result) manage the state transitions between entering a number, selecting an operator, and viewing the result.
3. **GUI Layout and Binding**: The layout uses the grid() geometry manager for precise, responsive placement of all buttons. lambda functions are used extensively to pass specific operator arguments to the generic operation() and numberEnter() methods.

:Screenshots / Results:

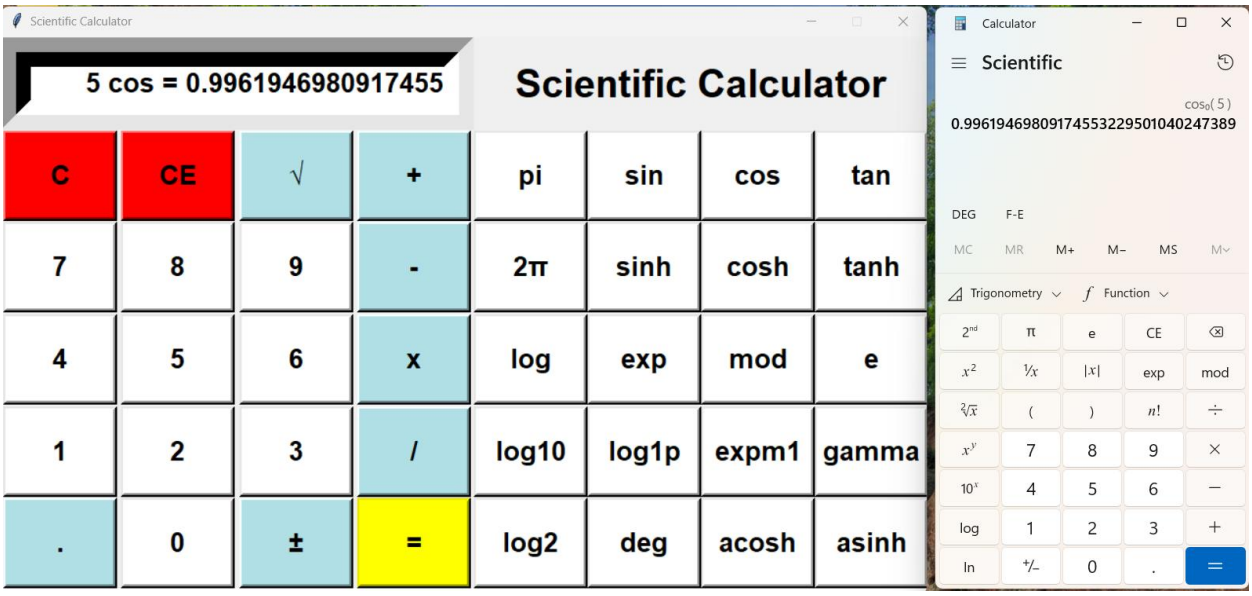
Error handling: for division 1/0 = Error.



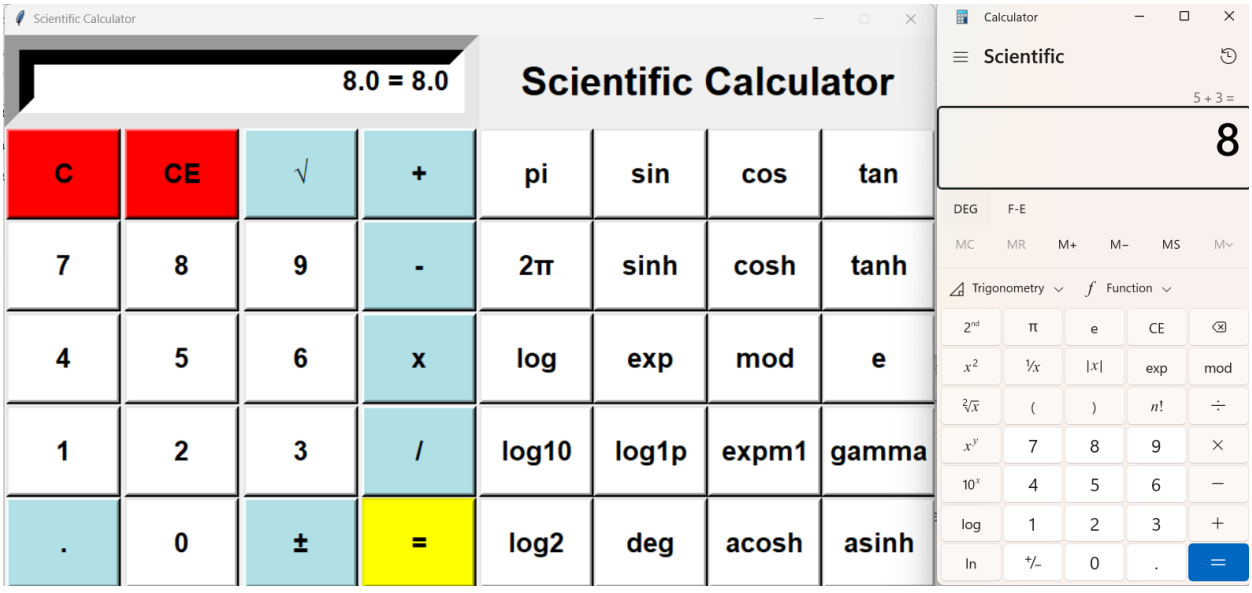
Square root of the number.



Input 5 and then cos it gives answer as 0.9961 approx.



Input 5 then + and then 3 gives the output as 8.



:Testing Approach:

The project utilizes Validation Tests to ensure the accuracy and stability of all functions:

Test Case	*Module*	*Expected Result*
Scientific Chaining	Scientific/Basic	Input 5 and then cos it gives answer as 0.9961 approx.
Zero Division	Error Handling	Input 1 / 0 should yield Error: Division by zero.
Input Reset	I/O Control	After 5 + 3 = 8.0, the next digit entered should start a new calculation, not append to 8.0.

Error Handling Test:

The try-except blocks in the core calculation methods are tested to confirm that the calculator displays a clean message and resets the state upon encountering math domain errors (e.g., log(0)) or runtime exceptions.

:Challenges Faced:

1. **State Management**: Managing the complex internal state of the calculator (when to reset self.current, when to set self.total, and whether an operation is pending using self.check_sum) was the primary challenge. This was resolved by meticulously defining the effects of the operation() and sum_of_total() methods on the state flags.
2. **Function Chaining (Scientific)**: Initially, scientific function results were only displayed, not saved to the state. This prevented chaining (sin(30) + 5). The solution involved refactoring all scientific methods to use self.calculate_and_update, which reliably sets the new result in self.total and self.current.

:Learnings & Key Takeaways:

- **Tkinter Mastery:** Gained practical experience in using grid() for complex layouts and lambda functions for efficient button binding.
- **OOP in GUI:** Reinforced the value of OOP by designing a centralized Calc class to manage the application's entire logical state, ensuring a modular and maintainable implementation[16].
- **State Machine Design:** Learned how to treat a functional calculator as a state machine, where every button press causes a controlled transition from one logical state to another (e.g., waiting for first operand $\rightarrow$ waiting for second operand).

:Future Enhancements:

The following features could be implemented to further enhance the project's complexity:

1. **Session Capture / Screen Recording:** Implement the conceptual code to use libraries like **Pillow** and **OpenCV** to capture the calculator window frames and stitch them into a video file upon pressing a "Stop" button, fulfilling the **Logging or Monitoring** non-functional requirement.
2. **History Log:** Implement a dedicated log window or display area to keep track of previous calculations performed in the session.
3. **Memory Functions:** Add M+, M-, MR, and MC buttons to store and recall numbers from a separate memory variable.
4. **Scientific Notation/Precision Control:** Add options to display results in scientific notation and allow the user to control the displayed decimal precision.

:References:

1. **Python Documentation:** Official Python documentation for the language features and syntax.
2. **Tkinter Documentation:** Reference materials for the Python standard GUI toolkit.
3. **Python math Module Documentation:** Used for implementing all trigonometric, exponential, and logarithmic functions.
4. **Build Your Own Project Instructions:** VITyarthi Project Guidelines (Source Document) .