Name: SHRADDHA ASHISH GULVE
Roll No: 25BAI11595
Course: First Year CSE
Submitted to: Monika vyas

1. Introduction

For this project, I created a scientific calculator using Python and Tkinter. It is a desktop app that can handle both basic calculations and scientific functions like sine, cosine, and logarithm. I wanted to develop something practical that I could use while also learning about GUI programming. The calculator has two modes: standard for everyday math and scientific for more complex calculations. It works offline and doesn't require any complicated setup.

2. Problem Statement

Most calculators have their downsides:

- Phone calculators are too basic.
- The Windows calculator is okay but somewhat annoying.
- Online calculators need an internet connection.
- Real calculators can be expensive.

I wanted to create something that:
- Offers both basic and scientific features.
- Is easy to use.
- Works offline.
- Is free and simple.

3. Requirements

What it needs to do:
- Add, subtract, multiply, and divide.
- Handle decimal numbers.
- Perform scientific functions (sine, cosine, tangent, logarithm, square root, exponential).
- Conduct power calculations (x^y).
- Display constants like pi and e.
- Include clear buttons.
- Switch between modes.
- Exit properly.

How it should work:
- Buttons should be large enough for easy clicking.
- The display should be clear.
- There should be no lag when clicking.
- It should run on Windows, Mac, and Linux.
- It should not crash if unusual inputs are entered.

4. System Design

The calculator has three main parts:
- GUI Part: All the buttons and display, created with Tkinter.
- Logic Part: The Calc class that handles all the calculations and keeps track of what is happening.

- Math Part: Python’s math module for performing the calculations.

The flow is straightforward: you click a button, the calculator updates, performs math if necessary, and shows the result.

5. Diagrams
Use Case
A user can:
- Enter numbers.
- Perform basic math (+, -, ×, ÷).
- Use scientific functions.
- Clear the display.
- Switch modes.
- Exit the app.

How it works:
Start App
▯
Show Calculator
▯
User clicks buttons
▯
Update display
▯
Perform calculation when the = button is pressed
▯
Show result
▯
Continue or Exit

Class Structure
Calc Class:
- total (stores the result)
- current (current number)
- op (current operation)
- some flags for tracking state

Methods:
- numberEnter() - for number button clicks
- operation() - for clicks on +, -, etc.
- sum_of_total() - when you press =
- All the scientific function methods

6. Why I Made These Choices
I chose Python and Tkinter because they are a simple way to create a GUI app. Tkinter comes with Python, so there’s no need for extra installation. The Calc class organizes all the logic, making it easier to understand compared to having everything scattered. I included two modes because not everyone needs the scientific functions, so the standard mode keeps it basic. The grid layout naturally fits the calculator and simplifies button placement. I used black buttons for numbers and blue for operations to make it easy to differentiate them.

7. How I Built It
I created a Calc class that manages all the calculator logic. It keeps track of:
- The current number being entered.

- The total from previous calculations.
- The current operation.
- Some Boolean flags to track the state.

For the GUI, I designed:
- A display using the Entry widget.
- Number buttons in a loop (this was cool to figure out).
- Operation buttons added one by one.
- A menu bar for switching modes.

The biggest issues I faced were:
- Chaining operations: When doing 5+3+2, it should show 8 after the second plus. This logic took time to figure out.
- Decimal points: I had to ensure you couldn't add multiple decimals in one number.
- Trigonometric functions: They gave incorrect answers because I forgot to convert degrees to radians.
- Button loop: When I created number buttons in a loop, they all performed the same function initially. I had to use lambda correctly.

8. Screenshots & Results
(add screenshots here)
Standard Mode: 480px wide, shows the basic calculator.
Scientific Mode: 944px wide, displays all the functions.
Tests I conducted:
- 5 + 3 = gives 8 □
- 90 sin = gives 1 □
- 2 x^y 3 = gives 8 □
- pi button = shows 3.14159... □
- Multiple decimals = only allows one □

9. Testing
I conducted manual testing with various calculations:
- Basic math operations are working fine.
- Scientific functions are functioning correctly.
- Switching modes works smoothly.
- Decimal numbers are handled well.
- The +/- button for negative numbers works.
- Constants like pi and e display correct values.

Problems I found:
- No keyboard support (only mouse).
- Division by zero might crash the app (didn't test this fully).
- Very long numbers get cut off on the display.
- No calculation history.

10. Problems I Faced
I struggled with understanding events, as it took time to realize how button clicks trigger functions. I needed to learn about lambda and callbacks. Keeping track of state—knowing the current number, what's stored, and the current operation—was confusing at first, so I drew some diagrams to help. Getting the grid layout right took multiple attempts. I kept mixing up row and column numbers. I also ran into issues with trigonometric accuracy; sin(90) returned the wrong answer because math functions use radians instead of degrees, so I had to add a conversion. Finally, I had to ensure that calculations like 5+3+2 worked correctly without just showing 3 repeatedly.

11. What I Learned

Technically, I learned how to create GUI applications, apply object-oriented programming, handle events and callbacks, use Python's math module, and manage layout in Tkinter.

In general, I realized the value of starting simple and gradually adding features (I built a basic calculator first and then added scientific functions). Testing often helps catch bugs early. Drawing layouts on paper before coding was beneficial. Documentation matters, including readme files and comments. Debugging takes time, but it’s a learning process.

I ended up spending more time on this project than I expected, but I learned a lot. Building something that actually works and looks decent is satisfying.

12. Future Improvements

Here are some features I want to add if I have time:

Easy ones:

- Keyboard support.
- Improved error handling.
- History of calculations.
- Memory buttons (M+, M-, etc.).

Harder ones:

- Graph plotting.
- Equation solving.
- Different themes.
- Unit conversions.

For now, the calculator does what it’s supposed to do, so I’m satisfied with it.

13. References

- Python documentation - https://docs.python.org
- Tkinter docs - https://tkdocs.com
- GeeksforGeeks tutorials
- Stack Overflow when I needed help
- YouTube videos on Tkinter

Conclusion

This was my first real project, and it taught me a lot. The calculator works well and has most of the features I wanted. There are definitely things I could improve, but it’s functional and I learned GUI programming, which was the goal. The biggest takeaway is that coding is an iterative process; the first version doesn’t have to be perfect, and you continue to improve. Testing is important, and user experience matters, even for simple projects. Overall, this was a valuable learning experience, and now I have a calculator I can actually use.