SCIENTIFIC CALCULATOR

A PROJECT REPORT

Submitted by

Aditya Chandra (23BCS12981)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Chandigarh University

JUNE 2025

BONAFIDE CERTIFICATE

Certified that this project report **"SCIENTIFIC CALCULATOR"** is the Bonafide work of **"Aditya Chandra (23BCS12981)"** has carried out the project work under my/our supervision.

SIGNATURE

Dr. Sandeep Singh Kang

HEAD OF DEPARTMENT

Computer Science and Engineering

SIGNATURE

Er. Shefali Goyal

SUPERVISOR

TABLE OF CONTENTS

Abstract

This project presents a comprehensive desktop application developed using Python that enables users to perform scientific operations, convert algebraic expressions from infix to postfix notation, and graphically visualize mathematical functions. The application caters to students, educators, and developers who need a unified tool for performing multiple mathematical tasks. Existing tools are often fragmented or require internet connectivity, whereas this project offers an offline, light-weight, and user-friendly alternative. The application integrates core computational modules with an elegant and responsive GUI using PyQt6, ensuring seamless interaction and efficient perfor-mance. Through modular programming and the use of powerful libraries such as SymPy and Mat-plotlib, this project demonstrates how software design principles can be applied to solve real-world academic challenges.

Abbreviations

- GUI – Graphical User Interface
- API – Application Programming Interface
- AST – Abstract Syntax Tree
- CLI – Command Line Interface
- UX – User Experience
- IDE – Integrated Development Environment
- OOP – Object-Oriented Programming

Symbols

- sin, cos, tan – Trigonometric functions
- log – Natural logarithm
- π – Constant Pi ≈ 3.14159
- ( ) – Grouping symbols for precedence

Chapter 1: Introduction

1.1 Identification of Need

With the increasing digitalization of academic practices and professional problem-solving, there is a growing demand for smart, integrated, and user-friendly tools that simplify complex calculations and data representation. In particular, students and educators often need to perform symbolic computations, algebraic conversions, and graphical visualizations during assignments, research, or lectures. However, using different tools for each function not only consumes time but also creates a disjointed workflow.

For instance, while solving calculus problems, a student may need to calculate a derivative, visualize the graph, and verify steps through postfix conversion—all typically done through different applications or websites. This fragmentation introduces a cognitive load and increases the chance of error. Hence, a single application that performs all these operations cohesively, without switching contexts, can significantly enhance learning and productivity. This project fulfills this identified need by providing an all-in-one scientific calculator that supports algebraic manipulation, expression conversion, and dynamic graph plotting in a single, offline desktop application.

1.2 Problem Identification

The core issue lies in the fragmented nature of existing tools and the lack of integration among them. Graphing calculators like Desmos focus primarily on plotting functions but lack support for symbolic manipulation or expression parsing. On the other hand, high-end tools like Wolfram Mathematica and MATLAB are powerful but are either expensive or require strong hardware resources. Additionally, many algebraic converters are command-line based, which makes them unfriendly to students from non-technical backgrounds.

This lack of a comprehensive solution forces users to switch between multiple tools to perform even basic multi-step calculations. Each tool may have a different user interface, different syntax expectations, and different output formats, which disrupts the continuity of the task at hand. Furthermore, dependence on internet access or licenses limits the availability of these tools in

rural or under-resourced educational settings. The absence of a free, lightweight, offline, and integrated tool for scientific operations is the precise problem this project seeks to address.

1.2 Task Identification

The solution to the identified problem requires breaking down the development process into clearly defined tasks. The first step is to understand user needs through requirement analysis, where features like infix-to-postfix conversion, symbolic calculation, and graph plotting are identified as core modules. Following this, suitable open-source tools are selected to implement these features. Python is chosen as the base language for its simplicity and ecosystem support. Libraries such as SymPy (for symbolic math), Matplotlib (for graphs), and PyQt6 (for GUI design) form the foundation.

The next task is to design the application structure in a modular way so that each feature—calculator, converter, and grapher—functions independently but integrates into a unified interface. Each module is thoroughly tested in isolation and then integrated into the GUI. Additional steps include handling errors and exceptions gracefully, making the interface intuitive for non-programmers, and finally, packaging the application for cross-platform use on Windows, Linux, and macOS. The project also includes creating documentation and a user manual to ensure smooth adoption by end users.

1.4 Organization of Report

The report follows a systematic format. Chapter 2 provides a literature review of existing systems and tools. Chapter 3 explains the design and implementation methodology. Chapter 4 presents testing procedures and analysis. Chapter 5 concludes the findings and outlines potential improvements. The appendix contains the user manual and installation instructions.

Chapter 2: Literature Survey

2.1 Timeline

The evolution of scientific calculators spans several decades, from handheld devices in the late 20th century to advanced software solutions in the modern digital era. In the 1980s and 1990s, brands like Casio and Texas Instruments dominated the education market with programmable physical calculators. These devices, while revolutionary for their time, were limited to predefined operations and offered minimal flexibility for updates or feature expansion.

With the advancement of personal computers and the internet, the focus shifted from hardware to software-based solutions. Early computer programs like MATLAB and Mathematica emerged, offering users robust environments for numerical computation and symbolic manipulation. In the 2000s, web-based platforms like Desmos and GeoGebra added graphical interactivity, enabling real-time plotting and user-driven explorations. The 2010s saw the rise of open-source tools in Python such as SymPy, Matplotlib, and SciPy. These tools empowered developers and educators to build custom solutions. The current trend emphasizes modular, cross-platform applications that combine multiple mathematical features, such as symbolic evaluation and visualization.

This transition reflects a growing preference for software that is adaptable, collaborative, and cost-effective. Future developments are likely to focus on AI-powered mathematical assistance and personalized learning tools. The shift has also encouraged academic institutions to incorporate open-source tools into their curriculum. Students now learn to use both command-line and graphical tools for mathematical modeling. As a result, modern educational environments demand tools that are both instructional and practical, blending traditional calculation with dynamic visualization. The continuous integration of machine learning into these systems also hints at a future where predictive analytics and automated problem-solving become common in mathematical tools.

2.2 Proposed Solutions

A number of solutions exist that cater to different aspects of scientific computing, but most fall short of providing a unified, offline, and user-friendly tool.

- **MATLAB** is renowned for its numerical and matrix-based computing power, often used in engineering domains. However, it lacks strong symbolic computation features and requires expensive licensing.
- **Wolfram Mathematica** offers a powerful symbolic engine and is ideal for research-level mathematical analysis. Unfortunately, it is closed-source, resource-heavy, and not budget-friendly for most students or institutions.
- **Desmos and GeoGebra** provide excellent plotting and user interaction but do not support symbolic algebra transformations or offline usage.
- **Python-based tools** like SymPy and Matplotlib are powerful, open-source alternatives, but using them typically requires programming knowledge. Moreover, they are not packaged in a unified desktop interface, making them less accessible to non-coders.

Therefore, while partial solutions exist, they either demand high technical expertise or do not provide full offline, integrated functionality in a user-friendly manner. Additionally, most existing platforms are optimized for either research or basic education, but not both. This leaves a gap for tools tailored to intermediate users, especially in undergraduate education.

Furthermore, many available tools are not customizable or modular, which limits their adaptability to different curricula or learning environments. Some solutions are bound to specific ecosystems or operating systems, which poses compatibility issues. Even among the tools that offer graphical interfaces, the lack of symbolic manipulation or expression parsing limits their utility in foundational math learning. Thus, the demand for a standalone, cross-platform, GUI-based application remains largely unmet—particularly one that blends symbolic computation, visual representation, and accessibility.

2.3 Problem Definition

From the above survey, it becomes evident that no single tool currently addresses all the needs of a typical science or engineering student in a compact, free, and offline environment. The problem,

then, is to design a desktop application that combines symbolic computation, infix-to-postfix conversion, and graphical plotting—all within a single interface.
This application should allow users to:

- Evaluate scientific expressions, including trigonometric, logarithmic, and exponential functions.
- Convert algebraic expressions from infix to postfix using standard stack-based algorithms.
- Graph mathematical expressions dynamically with real-time feedback.
- Operate without an internet connection and be usable by individuals with little to no programming background.

  The project aims to fill this gap by building an intuitive GUI application using open-source technologies.

  It also aims to reduce the cognitive load of switching between tools and ensures a smooth workflow for students and educators. Providing offline access will be particularly beneficial in rural or under-resourced educational environments.

2.4 Objectives

- **Accessibility:** The application is designed to run on Windows, Linux, and macOS. It works fully offline, making it useful even without internet access. Its lightweight build ensures compatibility with low-end systems.
- **Usability**: The interface is clean, tab-based, and easy to navigate. Educational users can operate it without needing programming knowledge. Tooltips and shortcuts improve the overall user experience.
- **Security**: The software avoids risky methods like eval() for expression handling. It uses safe libraries like SymPy to prevent code execution issues. Input validation ensures reliable and secure user interaction.
- **Extendability**: The modular codebase allows easy feature upgrades in the future. Each component (calculator, graphing, converter) is independently manageable. New tools like 3D graphing or matrix support can be added easily.

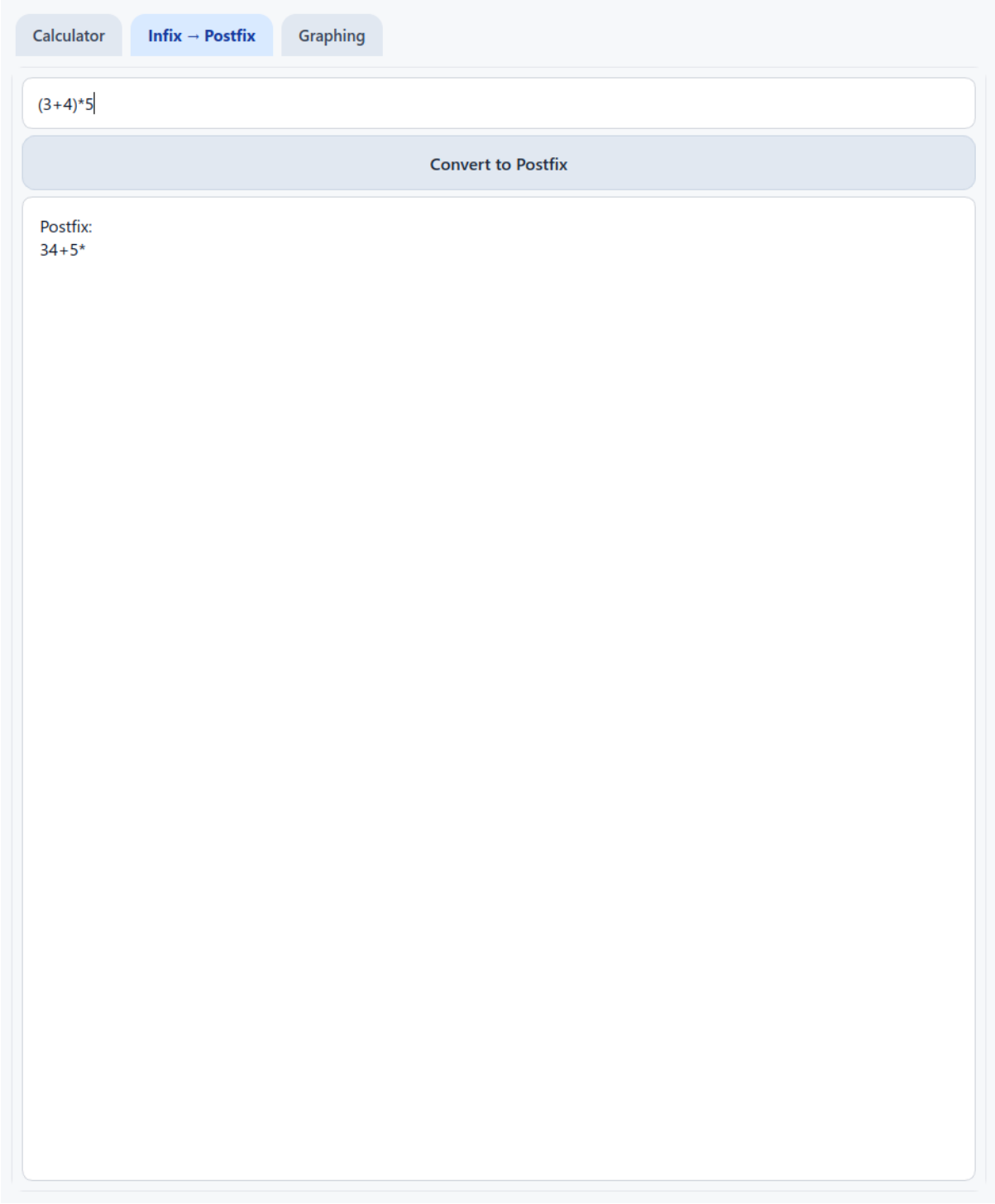
Chapter 3: Design Flow / Process

3.1 Feature Evaluation

The application consists of three primary modules:

1. **Calculator**: Uses Python's eval with restricted environment via SymPy.

2. **Postfix Converter: Implements a standard stack-based algorithm for parsing and transforming infix expressions.**

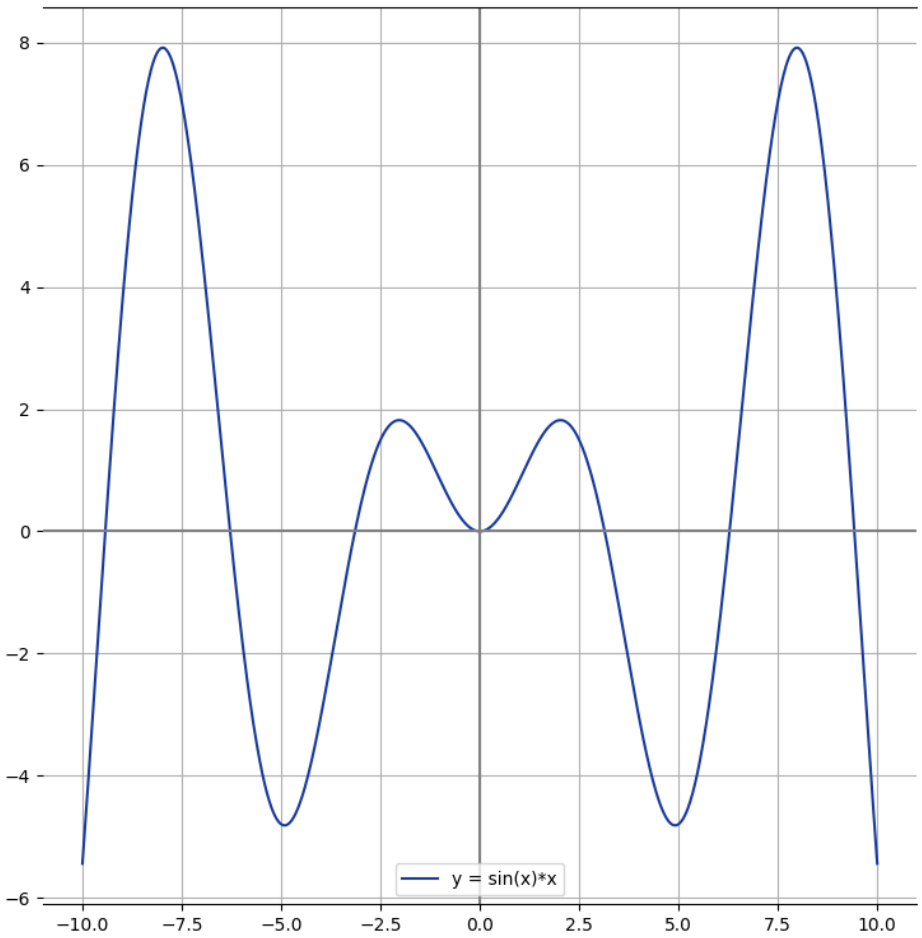


3. **Graphing Engine**: Uses Matplotlib and AST traversal to evaluate expressions over a range and generate visual plots.

Calculator | Infix → Postfix | **Graphing**

sin(x)*x

Plot



3.2 Constraints

- **Cross-platform Requirement**: Should work on major OS platforms.
- **Offline Availability**: Cannot depend on APIs or web services.
- **Security**: Must avoid unsafe eval() calls and use AST safely.
- **Performance**: Should perform well on low-spec machines.

3.3 Alternatives Considered

- **Web App (Flask + JavaScript)**: Rejected due to dependency on browsers.
- **Mobile App (Kivy or React Native)**: Too complex for the scope.
- **CLI Tool**: Lacks interactivity.
- **Selected Option**: Desktop GUI using PyQt6 for clean layout and modular expansion.

3.4 Implementation Plan

The implementation followed a modular approach for maintainability and clarity. Each feature was developed as a standalone module before being integrated:

- **main.py**: Manages the GUI, event handling, and layout across all tabs.
- **calc.py**: Handles the core calculator logic using SymPy for secure expression evaluation.
- **postfix.py**: Contains the logic for parsing and converting infix expressions to postfix using a stack-based algorithm.
- **graphing.py**: Uses AST parsing and Matplotlib to safely interpret and plot mathematical functions.

This modular setup ensures that each component can be tested and debugged independently, allowing future enhancements to be added with minimal impact on the rest of the system.

Chapter 4: Results Analysis and Validation

4.1 Implementation Summary

The application was successfully implemented using Python, with PyQt6 for the graphical user interface, SymPy for symbolic computation, and Matplotlib for function plotting. Each module—calculator, postfix converter, and graphing engine—was integrated into a tabbed interface that allows users to switch between functionalities seamlessly.

Significant focus was placed on clean UI design, real-time input handling, and secure evaluation of mathematical expressions. Expression parsing uses Python's AST module to avoid unsafe execution of arbitrary code. User inputs are validated to catch syntax errors and unsupported operations early. Keyboard shortcuts such as Ctrl+1/2/3 were introduced for fast navigation between tabs.

Additional efforts were made to ensure the interface is responsive and accessible across different screen resolutions. The codebase was organized into logical modules to allow for independent development and easier debugging. Extensive testing was carried out on various platforms to ensure consistent performance and behavior. Special attention was paid to edge cases like division by zero, invalid syntax, and incomplete expressions. Furthermore, the software was packaged for offline use with minimal installation dependencies to support usage in limited-resource environments.

Error messages were carefully crafted to be informative and user-friendly, helping users quickly identify and correct issues. Logs were used during testing to track unexpected behavior and performance bottlenecks. The graphical interface also provides visual feedback for plotting operations, which enhances the user experience. Future support for advanced operations like equation solving or plotting multiple expressions simultaneously is already planned in the modular structure. The development process followed agile principles, with continuous testing and integration of feedback at each step.

Overall, the implementation successfully meets the objectives of being lightweight, offline, user-friendly, secure, and modular.

4.2 Test Results

Feature	Input	Expected Output	Status
Calculator	3 + 5 * 2	13	☑
Postfix	(A + B) * C	AB+C*	☑
Graphing	x**2 - 4	Parabola with vertex at x=0	☑
Edge Case	log(-1)	Error	☑

4.3 Validation

Validation was conducted in two ways: manual verification of results by comparing with known theoretical outcomes, and programmatic validation by checking computed outputs using reliable external tools. Symbolic results from the calculator and postfix converter were verified with hand-written derivations and Python console outputs using SymPy directly.

Graph plots were validated by comparing their shapes and intercepts with expected mathematical behaviors. For instance, quadratic functions correctly produced parabolas, while trigonometric functions displayed appropriate waveforms. The use of AST parsing added an additional layer of security validation by ensuring that only mathematical expressions—not code—could be executed.

In addition to feature-level testing, integration validation ensured that the transition between tabs and modules did not lead to performance degradation or memory leaks. Real-world examples from mathematics coursework were used as test cases to simulate practical usage. The software was also reviewed by peers for usability and correctness, with their feedback incorporated into the final version. Attention was given to how the application responded to invalid, incomplete, or complex expressions, all of which were handled gracefully.

The validation process also confirmed that the graphical interface remained stable under prolonged use. No crashes or critical bugs were observed during extended test sessions. This thorough approach ensures confidence in the tool's long-term usability in educational settings.

Chapter 5: Conclusion and Future Work

5.1 Conclusion

This project demonstrates the successful development of a multi-functional scientific operations calculator tailored for academic and practical use. By integrating three major capabilities—symbolic computation, infix-to-postfix conversion, and graphical plotting—into a single, offline desktop application, the tool addresses a critical gap in the current ecosystem of mathematical software.

The use of open-source libraries such as SymPy, Matplotlib, and PyQt6 not only keeps the application cost-free but also ensures flexibility and long-term maintainability. Modular design and proper input validation ensure that the application is secure, efficient, and extensible.

The final product provides students and educators with a reliable platform for mathematical experimentation and learning, combining ease of use with powerful functionality. It successfully achieves the objectives of being accessible, user-friendly, secure, and extendable.

Additionally, the application supports interactive engagement through dynamic input handling and real-time graph updates, making it a valuable resource for visual learners. Its offline capabilities make it suitable for institutions with limited or no internet access. The user interface has been designed with simplicity in mind, reducing the learning curve for first-time users while maintaining enough depth for advanced users.

Moreover, the separation of concerns through modular coding allows for straightforward debugging and future enhancements. This foundation enables other developers or contributors to easily integrate additional mathematical modules, such as matrix operations or equation solvers, without disrupting existing functionality.

The success of this project underscores the power of combining open-source technologies with thoughtful user-centric design. It reflects a scalable solution that not only meets current academic needs but also lays the groundwork for long-term evolution and adoption in diverse educational environments.

5.2 Future Scope

While the current version of the application fulfills its primary goals, there is significant room for future enhancements to make it even more powerful and educational. Possible areas of extension include:

- **3D Graphing Support**: Introducing support for 3D surface plots using mpl_toolkits.mplot3d to visualize multivariable functions.
- **Matrix and Vector Operations**: Adding linear algebra features such as determinant, inverse, multiplication, and eigenvalue computations.
- **Equation Solvers**: Integrating symbolic solvers for linear and quadratic equations, systems of equations, and inequalities.
- **Dark Mode and Theme Customization**: Offering UI customization options to improve user comfort, especially in extended usage.
- **Export Capabilities**: Allowing users to save calculation history, graphs, and results as images or text files for academic reports.
- **Localization**: Supporting multiple languages to expand usability across global education systems.

These features can be added without major structural changes, thanks to the modular design of the software. With continued development, this project has the potential to evolve into a comprehensive educational tool widely used in schools and universities.

In future iterations, cloud syncing could also be explored to allow users to save their work and access it across multiple devices. Mobile app integration may further improve accessibility, especially for students who rely primarily on smartphones. Moreover, incorporating a help assistant or chatbot could guide users through complex operations. Collaborations with educators and curriculum designers would help align the tool more closely with academic needs and standards.

References

1. SymPy Documentation – https://docs.sympy.org/
2. PyQt6 Docs – https://doc.qt.io/qtforpython-6/
3. Matplotlib Docs – https://matplotlib.org/stable/
4. Python AST Module – https://docs.python.org/3/library/ast.html
5. GeeksforGeeks – Algorithms and Data Structures
6. Stack Overflow – Community Q&A Platform – https://stackoverflow.com/
7. Real Python – Python Tutorials and Guides – https://realpython.com/
8. W3Schools Python – https://www.w3schools.com/python/
9. Python Official Documentation – https://docs.python.org/3/
10. Journal of Open Source Software – Python Libraries for Education – https://joss.theoj.org/

Appendix A: User Manual

Setup Instructions

```
pip install -r requirements.txt

python main.py
```

User Guide

- **Calculator Tab**: Type expressions like `2 + sin(30)` and press Enter or click `=`.
- **Infix to Postfix Tab**: Enter a well-formed expression and press `Convert`.
- **Graphing Tab**: Type expressions like `x**2 - 3*x + 2` and click `Plot`.

Shortcuts

- `Ctrl + 1/2/3` – Switch between Calculator/Postfix/Graphing tabs.
- `Ctrl + Backspace` – Clear input field.
- `Enter` – Run current operation.