

Scientific Calculator Overview

Application Overview

The scientific calculator is a GUI-based desktop application that provides both basic arithmetic and advanced mathematical functions. The application features a modern dark-themed interface built with CustomTkinter, offering an enhanced user experience compared to traditional Tkinter applications.

Key Features

Feature Category	Description	Implementation
GUI Framework	Modern, dark-themed interface	CustomTkinter >=5.2.0
Mathematical Engine	Scientific computing capabilities	NumPy >=1.24.0
Expression Evaluation	Safe mathematical expression processing	Python eval with validation
Visual Theming	Configurable dark mode styling	JSON-based theme system
History Management	Calculation result tracking	Built-in memory system

Runtime Environment

The application requires Python 3.10 or higher and utilizes modern Python features for enhanced performance and maintainability. The application is designed as a standalone desktop application that launches directly through the `main.py` entry point.

Application Initialization

The application follows a standard initialization pattern:

- Theme Configuration** - Sets CustomTkinter to dark mode and loads custom theme
- Application Instantiation** - Creates **CalculatorApp** instance from calculator package
- Event Loop Execution** - Starts the CustomTkinter main event loop

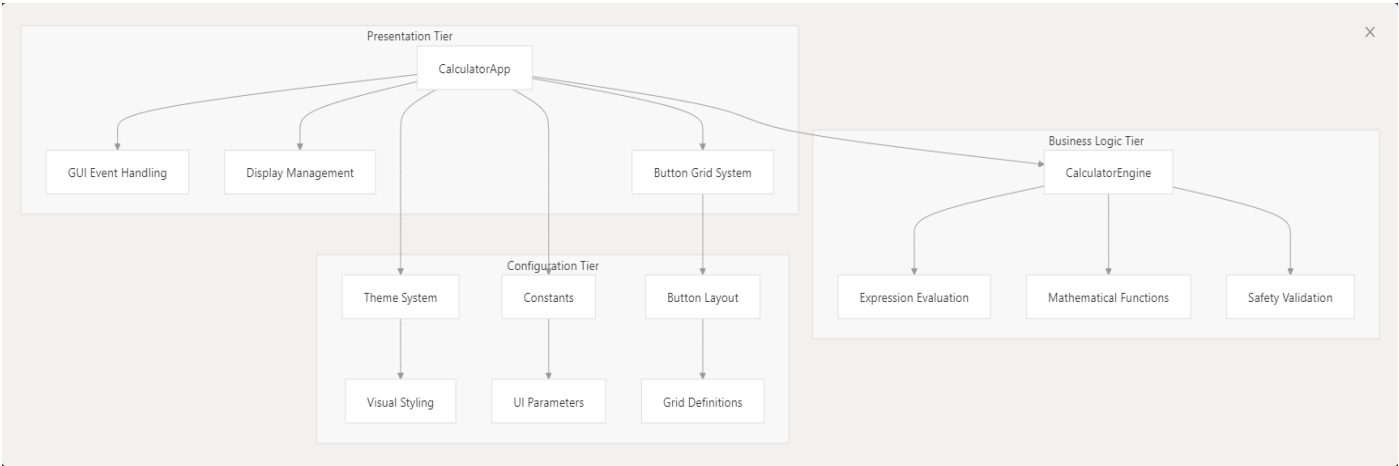
System Requirements

Component	Requirement	Purpose
Python Version	>=3.10	Modern language features and performance
CustomTkinter	>=5.2.0	Modern GUI framework with theming
NumPy	>=1.24.0	Scientific computing and mathematical operations

The scientific calculator represents a well-architected desktop application that balances modern UI design with robust mathematical functionality, providing users with both an aesthetically pleasing interface and powerful computational capabilities.

System Architecture Overview

High-Level Component Structure



Getting Started

Installation Process

The project uses **uv** as the dependency manager and can be installed using standard Python packaging tools. The installation process involves setting up the Python environment and installing the required dependencies.

Installation Steps

1. **Clone the repository and navigate to the project directory**
2. **Ensure Python 3.12 is available** (or Python ≥ 3.10) The project specifies Python 3.12 in
3. **Install using uv (recommended)**

```
uv sync
```

4. **Alternative: Install using pip**

```
pip install -e
```

The dependency resolution is managed by [uv.lock1-181](#) which ensures reproducible builds across different environments.

Running the Application

The calculator application is launched through the main entry point which initializes the GUI framework and starts the application loop.

Launch Commands

Standard execution:

```
python main.py
```

Using uv:

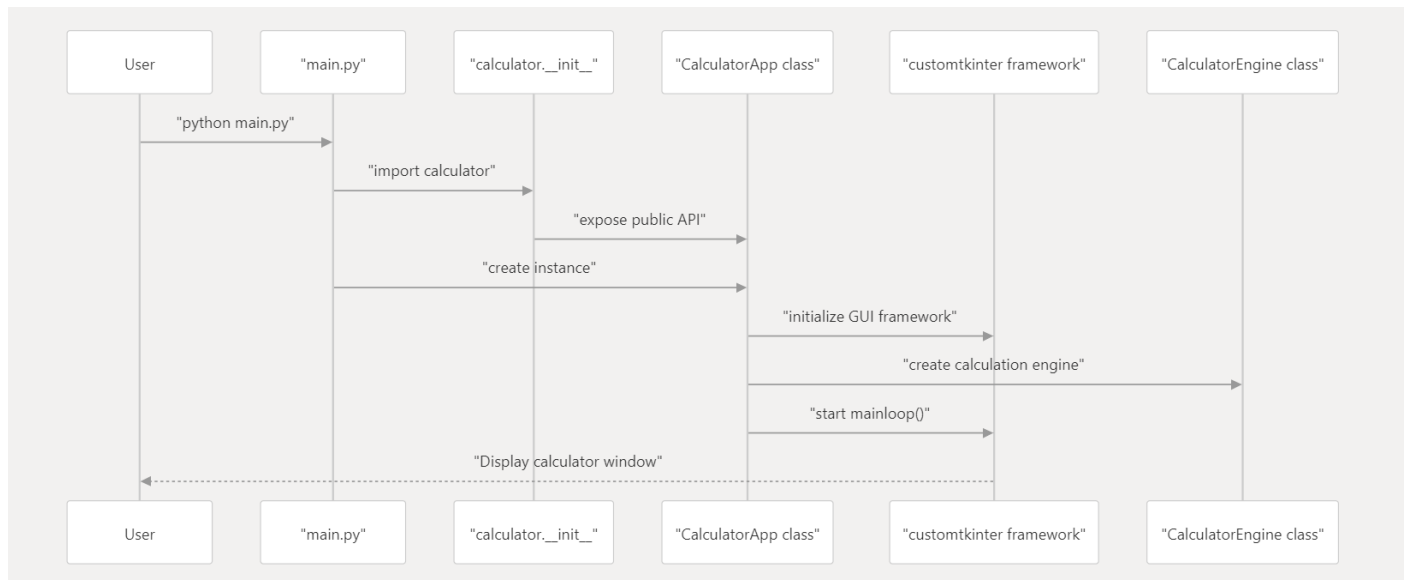
```
uv run python main.py
```

The main entry point is located at [main.py](#) and initializes the entire application stack including theme loading and GUI setup.

Project Structure Overview

The codebase follows a modular structure with clear separation between GUI components, calculation logic, and configuration.

Application Startup Sequence



Verification Steps

After successful installation and launch, verify the application is working correctly:

Basic Functionality Test

1. **GUI Window Loads:** The CustomTkinter-based calculator window should appear
2. **Button Grid Displays:** Mathematical operation buttons and number pad are visible
3. **Display Field Active:** Expression input and result display areas are functional
4. **Theme Loading:** Dark mode styling should be applied per [calculator/theme.json](#)

Quick Calculation Test

Perform a simple calculation to verify the **CalculatorEngine** integration:

- Enter: **2 + 3**
- Expected result: **5**
- This tests the basic expression evaluation pipeline

Scientific Function Test

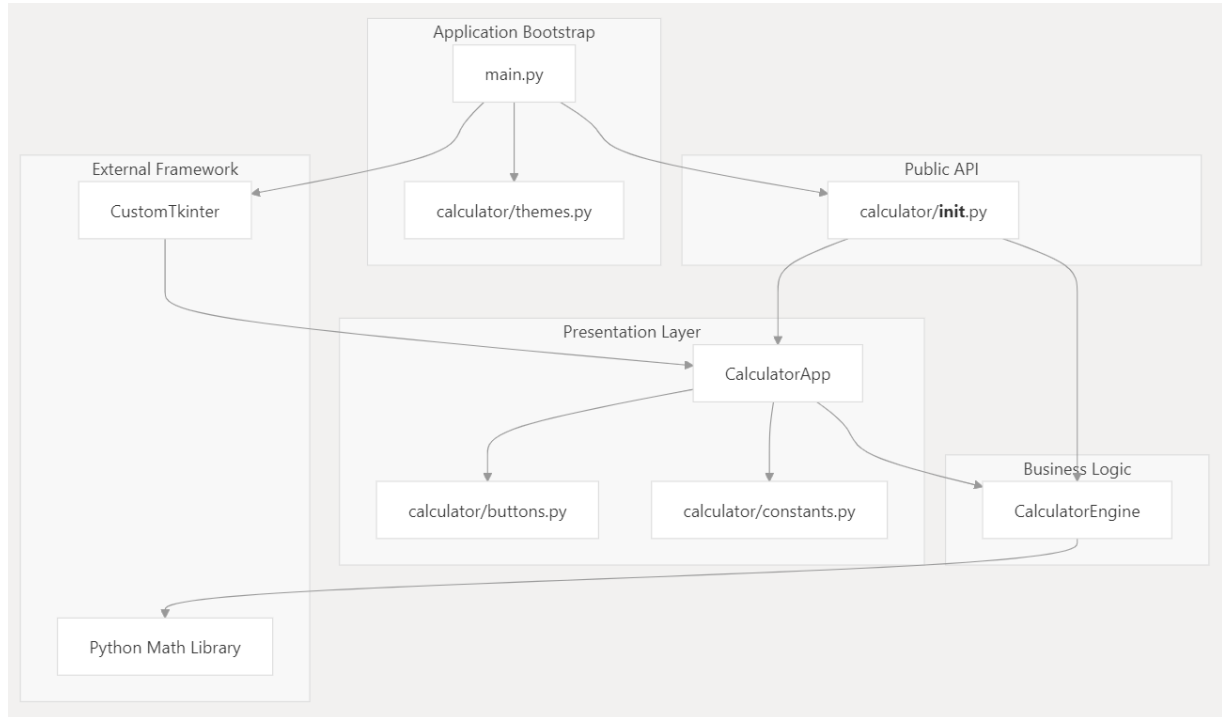
Test advanced mathematical capabilities:

- Enter: **sin(30)**
- Expected result: Should compute trigonometric function using NumPy backend
- This verifies NumPy integration and scientific computing features

If any verification step fails, check that all dependencies from [uv.lock](#) are properly installed and that the Python version matches [.python-version1](#)

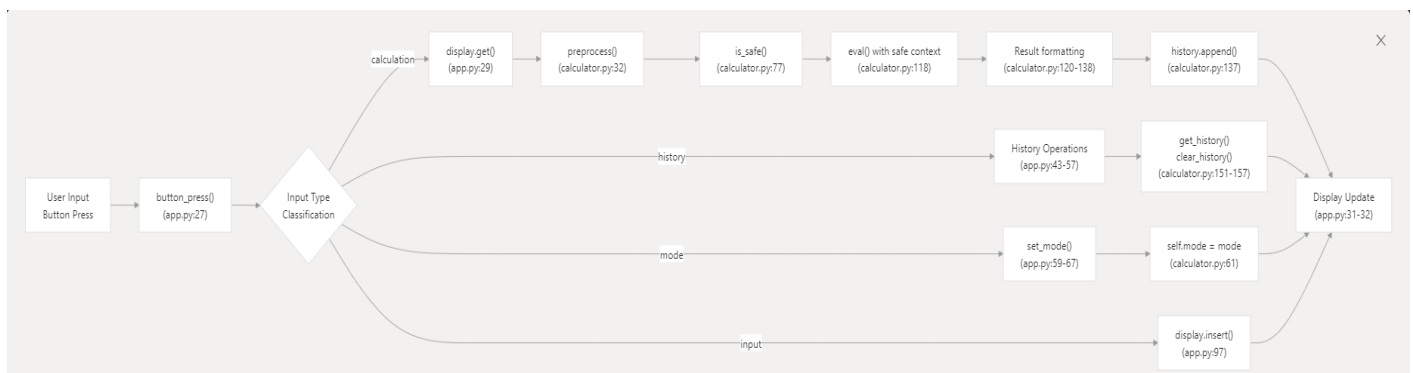
System Architecture

The calculator application follows a layered architecture with clear separation between presentation, business logic, and configuration concerns. The system is bootstrapped through a single entry point that configures the GUI framework and launches the main application window.



Data Flow Architecture

Mathematical expressions flow through a multi-stage processing pipeline that ensures both security and accuracy.



Data Flow: Expression Processing Pipeline

Mathematical expressions undergo preprocessing for notation conversion, safety validation to prevent code injection, secure evaluation using a restricted context, and result formatting before display and history storage.

Key Architectural Principles

Principle	Implementation	Code Evidence
Separation of Concerns	GUI logic separated from calculation logic	CalculatorApp class vs CalculatorEngine class
Single Responsibility	Each class has one primary purpose	CalculatorEngine handles math, CalculatorApp handles UI
Security by Design	Expression validation prevents code injection	is_safe() method blocks dangerous constructs
Event-Driven Architecture	User interactions trigger method calls	button_press() method dispatches events
Modular Design	Functionality split across focused modules	Separate files for app, calculator, buttons, themes
Configuration Externalization	UI settings and themes kept separate	Constants and theme files separate from logic

Architectural Principles: Design Patterns and Implementation Strategies

The codebase demonstrates clean architecture principles with clear boundaries between layers, secure evaluation practices, and modular organization that supports maintainability and extensibility.

Core Components

Purpose and Scope

This document provides an overview of the main functional components that comprise the scientific calculator application. It describes the primary classes, modules, and their interactions within the system architecture. For detailed implementation specifics of individual components, see the [Calculator Engine \(4.1\)](#), [GUI Application \(4.2\)](#), [User Interface Components](#), and [Theming System](#) sections.

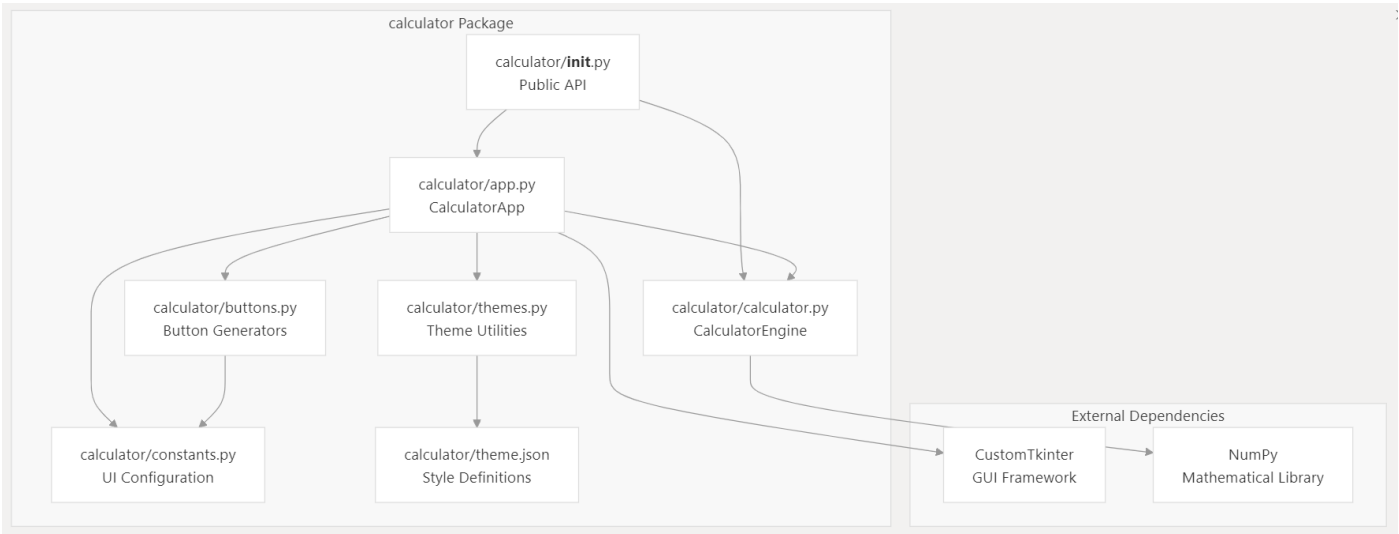
Component Architecture Overview

The calculator application is built around four core components that work together to provide a complete scientific calculator experience:

Component	Primary Module	Key Class	Responsibility
Calculation Engine	<code>calculator/calculator.py</code>	<code>CalculatorEngine</code>	Mathematical expression evaluation and computation
GUI Application	<code>calculator/app.py</code>	<code>CalculatorApp</code>	Main application window and event coordination
UI Components	<code>calculator/buttons.py</code>	Button generation functions	Interactive interface elements and layout
Theming System	<code>calculator/themes.py</code>	Theme utilities	Visual styling and appearance configuration

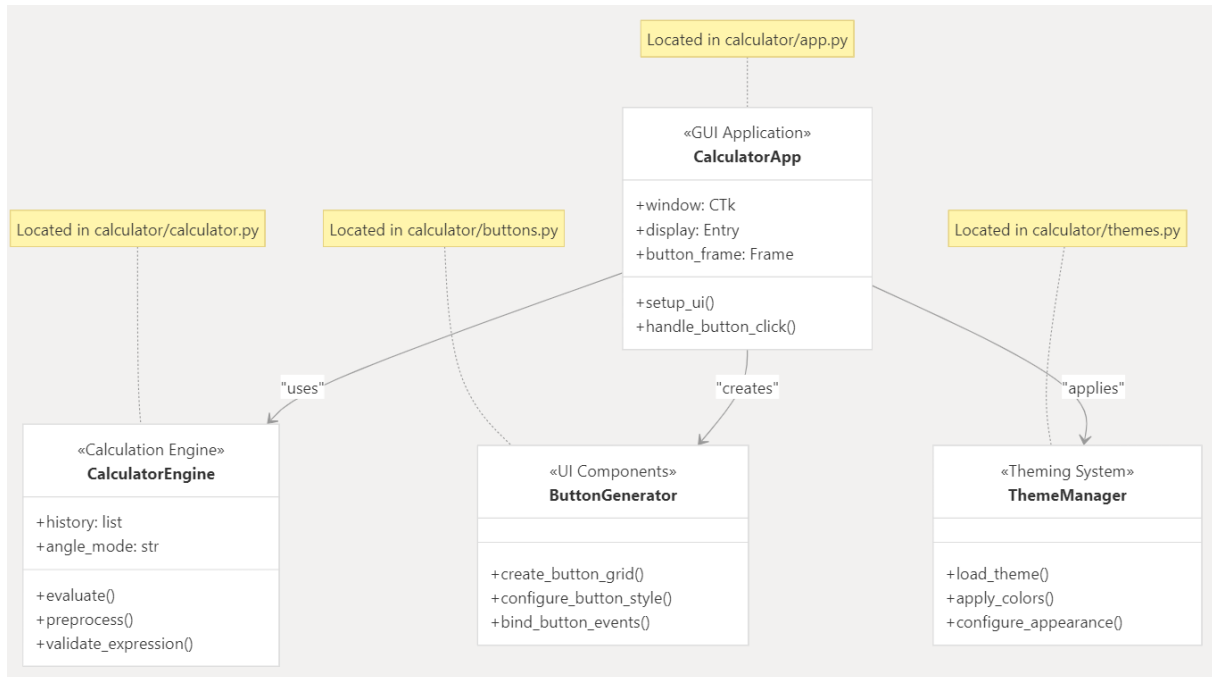
Component Interaction Architecture

The following diagram shows how the core components interact with each other at the class level:



Core Component Mapping

This diagram maps the natural language component names to their actual code entities:



Component Responsibilities

CalculatorEngine

The **CalculatorEngine** class serves as the mathematical computation core, handling:

- Expression parsing and preprocessing
- Safety validation of mathematical expressions
- Scientific function evaluation using NumPy
- Calculation history management
- Angle mode management (degrees/radians)

CalculatorApp

The **CalculatorApp** class manages the graphical user interface:

- Main application window creation using CustomTkinter
- Event handling for user interactions
- Display updates and visual feedback
- Integration between UI components and calculation engine
- Application lifecycle management

UI Components

The UI component system encompasses:

- Dynamic button grid generation
- Layout management and responsive design

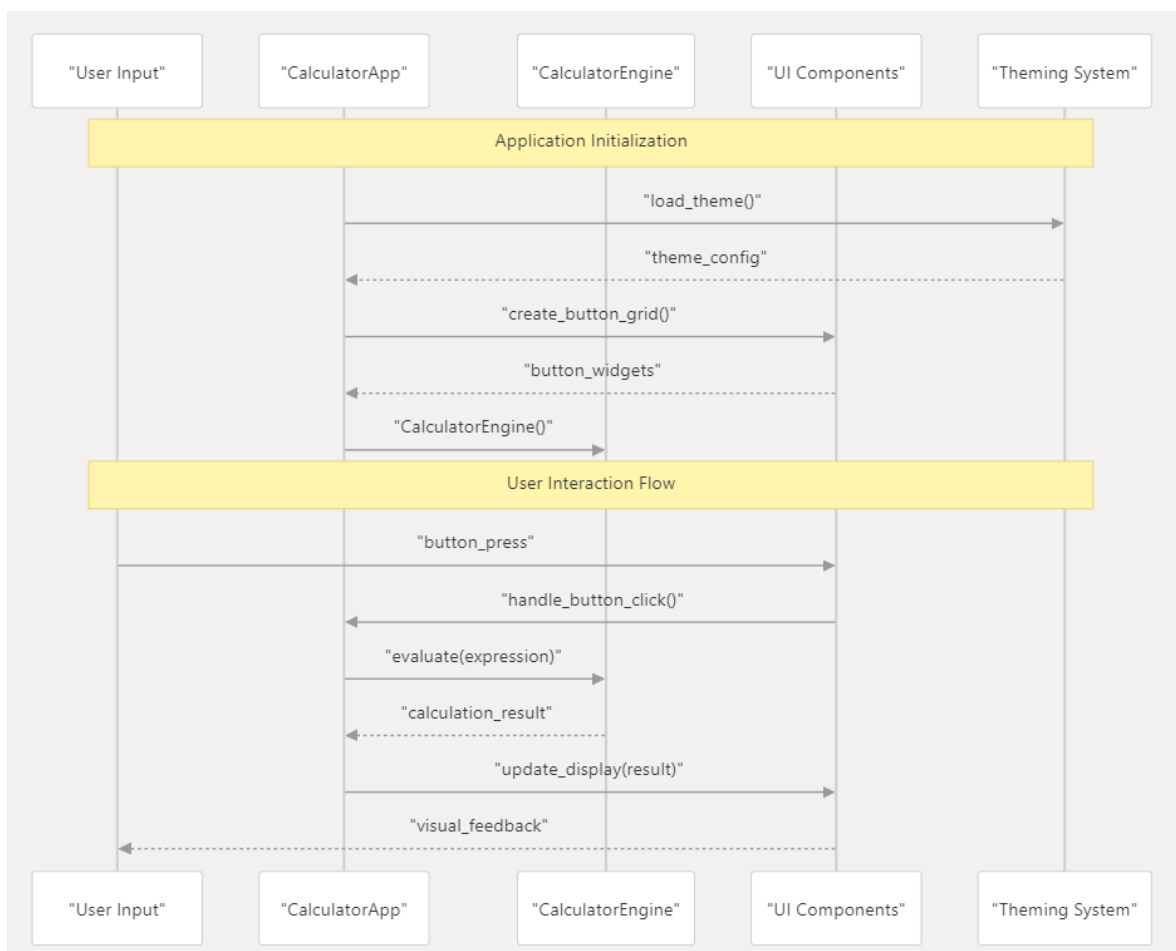
- Button styling and event binding
- Display field configuration
- UI constants and configuration parameters

Theming System

The theming system provides visual customization:

- JSON-based theme configuration loading
- Dark mode and light mode support
- CustomTkinter widget styling
- Color scheme application
- Theme path resolution and management

Data Flow Between Components



Calculator Engine

The Calculator Engine is the core computational component that handles mathematical expression evaluation, input preprocessing, safety validation, and calculation history management. It provides a secure and robust interface for evaluating complex mathematical expressions while maintaining a comprehensive history of calculations.

Class Overview

The **CalculatorEngine** class serves as the central processing unit for all mathematical operations in the calculator application. It encapsulates expression evaluation logic, maintains calculation state, and provides safety mechanisms to prevent code injection attacks.

Expression Processing Pipeline

The calculator engine processes user input through a multi-stage pipeline that transforms calculator notation into evaluable Python expressions, validates security constraints, and produces formatted results.

Preprocessing Transformations

The **preprocess** method performs several key transformations to convert user-friendly calculator notation into Python-evaluable expressions:

Input Notation	Transformation	Python Expression
$\sqrt{25}$	Square root symbols	<code>sqrt(25)</code>
$\sqrt[3]{8}$	Cube root symbols	<code>cbrt(8)</code>
2^3	Exponentiation	<code>2**3</code>
π	Pi constant	<code>pi</code>
$5!$	Factorial	<code>fact(5)</code>
2π	Implicit multiplication	<code>2*pi</code>
$\sin(30)$	Trigonometric (DEG mode)	<code>sin((30)*pi/180)</code>

Result Formatting and Error Handling

The engine provides comprehensive result formatting and error handling to ensure user-friendly output across different numeric types and error conditions.

Result Type Handling

Result Type	Formatting Logic	Example
Complex numbers	Real + imaginary parts	3.0+4.0i
Near-integers	Convert to integer	5.0000000001 → 5
Very small numbers	Convert to zero	1e-12 → 0
Very large numbers	Scientific notation	1e12 → 1.000000e+12
Standard floats	10 significant digits	3.141592654

Error Handling

Comprehensive Error Management

The calculator implements robust error handling for various scenarios:

Error Types Handled:

- **Division by Zero:** Prevents mathematical undefined operations
- **Value Errors:** Catches domain errors (e.g., square root of negative numbers)
- **Overflow Errors:** Handles numbers too large for computation
- **Syntax Errors:** Catches malformed expressions
- **General Exceptions:** Fallback for unexpected errors

Input Validation

Before evaluation, expressions are checked for safety:

This prevents:

- **Code Injection:** Blocks dangerous Python code execution
- **File System Access:** Prevents unauthorized file operations
- **System Calls:** Blocks system-level command execution

This document covers the **CalculatorApp** class, which serves as the main graphical user interface for the scientific calculator application. It handles window management, user interface layout, event processing, and integration with the calculation engine. For information about the underlying calculation logic.

Overview

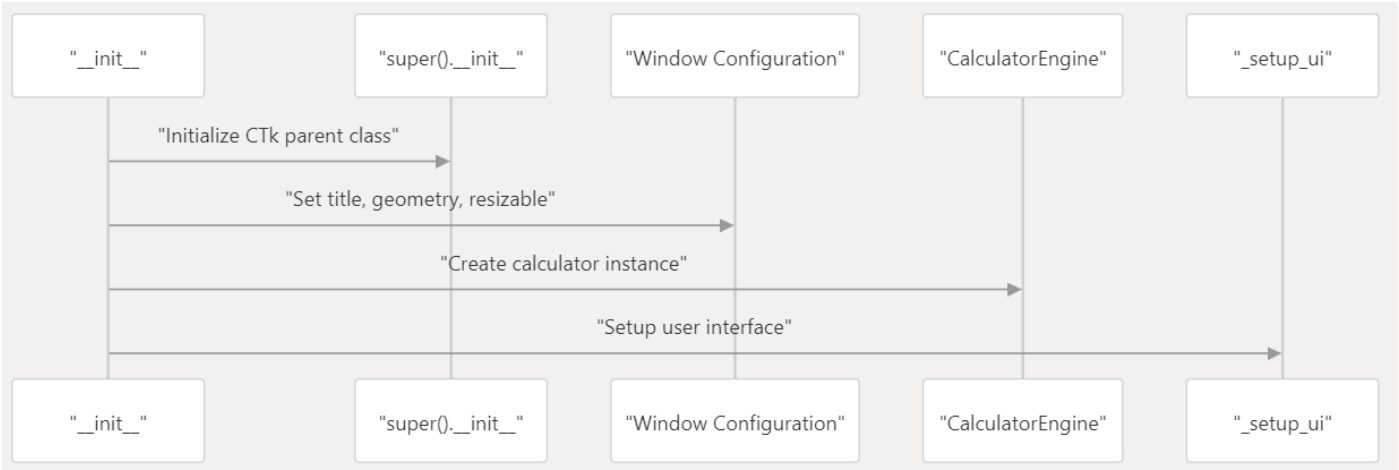
The **CalculatorApp** class extends CustomTkinter's **CTk** class to create the main application window. It provides a complete GUI framework that manages user interactions, displays mathematical expressions and results, and coordinates with the calculation engine to perform computations.

Application Initialization

The **CalculatorApp** constructor performs essential setup tasks including window configuration, calculator engine instantiation, and UI initialization.

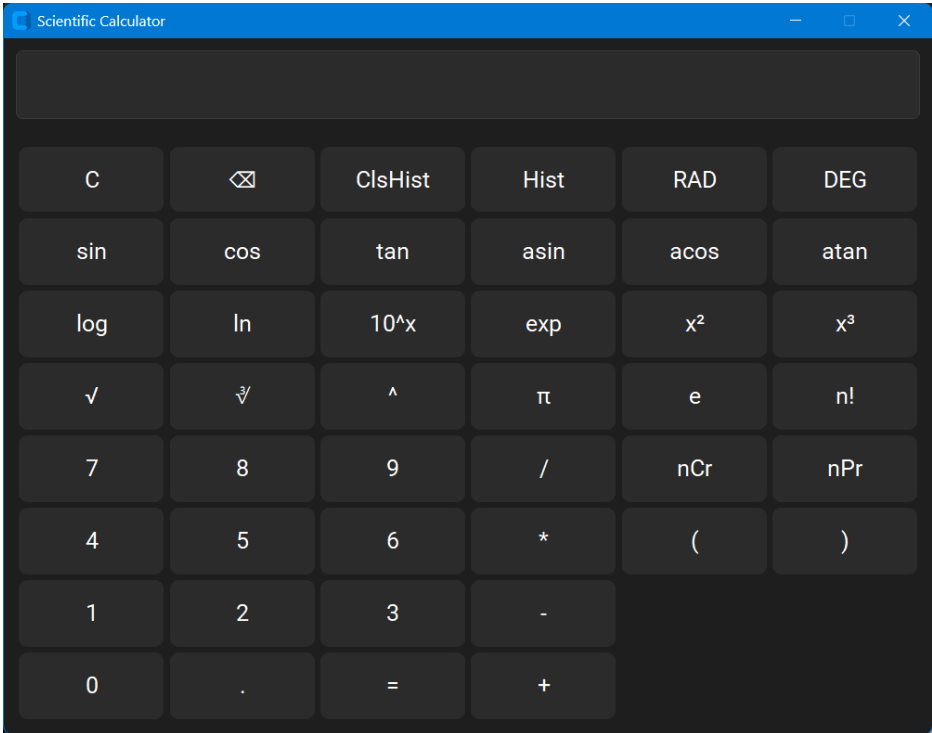
Component	Configuration	Purpose
Window Title	"Scientific Calculator"	Identifies the application
Window Size	WIDTH × HEIGHT from constants	Sets fixed window dimensions
Resizable	False	Prevents window resizing
Calculator Engine	CalculatorEngine() instance	Provides calculation functionality

The initialization process follows this sequence:



User Interface Layout

The UI layout uses a two-row grid system with the display field at the top and the button grid below.



Display Component

The display is implemented as a **CTkEntry** widget with the following specifications:

Property	Value	Purpose
Font	("Roboto Mono", 30)	Monospace font for number alignment
Justify	"right"	Right-align text like traditional calculators
Height	60 pixels	Adequate space for expressions
Grid Position	Row 0, Column 0	Top of the application
Sticky	"nsew"	Expand to fill available space

Display Management

The display component serves dual purposes as both an input field for mathematical expressions and an output area for results and system messages.

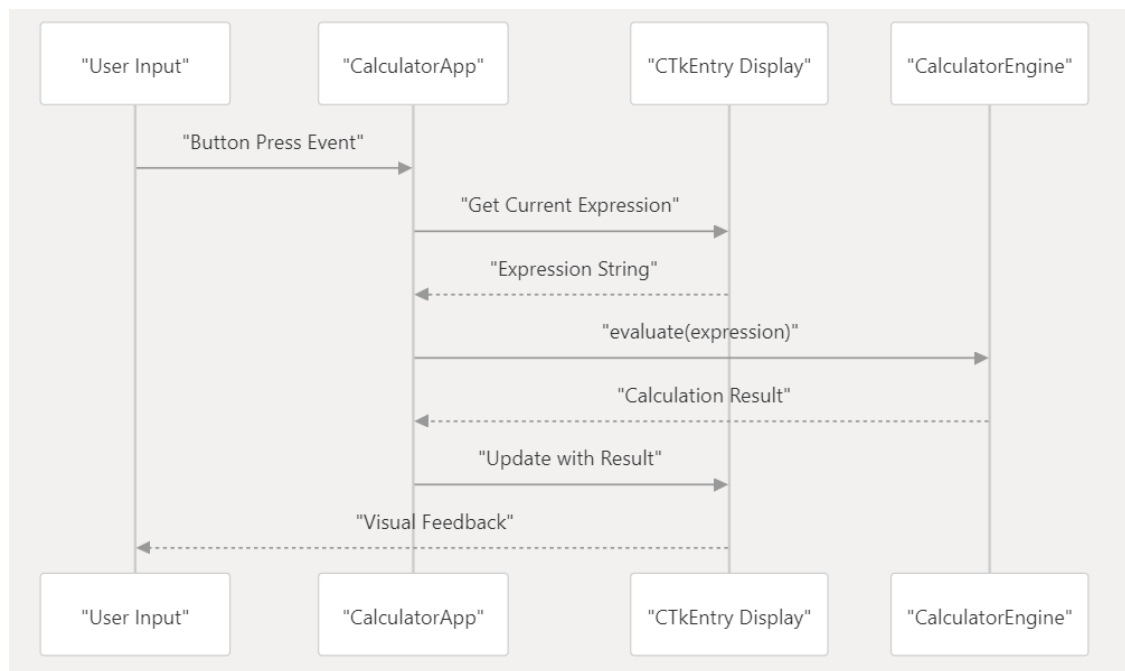
Display State Management

The GUI manages display content through three primary operations:

1. **Content Retrieval:** `self.display.get()` - Retrieves current display text
2. **Content Clearing:** `self.display.delete(0, "end")` - Removes all display content
3. **Content Insertion:** `self.display.insert(position, text)` - Adds text at specified position

Calculator Engine Integration

The GUI application maintains a composition relationship with the **CalculatorEngine**, delegating all computational tasks while managing the user interface state.



Engine Method Utilization

Engine Method	GUI Usage	Trigger Event
<code>evaluate(expression)</code>	Process mathematical expressions	"=" button press
<code>get_history()</code>	Retrieve calculation history	"Hist" button press
<code>clear_history()</code>	Remove all history entries	"ClsHist" button press
<code>get_last_result()</code>	Retrieve previous result	"LastAns" button press
<code>set_mode(mode)</code>	Switch DEG/RAD modes	"DEG"/"RAD" button press


User Interface Components

This page documents the user interface components system that defines the calculator's interactive elements, including the button creation system, layout management, and UI configuration constants. The components covered here handle the visual presentation and user input capture for the calculator application.

Button Grid System

The calculator's user interface is built around a structured button grid that organizes all interactive elements in a logical layout. The button grid is defined as a two-dimensional array that maps the physical layout of the calculator interface.

Button Types and Functions

Category	Buttons	Purpose
Clear Operations	C,  , CIsHist	Clear display, backspace, clear history
Mode/History	Hist, RAD, DEG	View history, toggle angle modes
Trigonometric	sin, cos, tan, asin, acos, atan	Basic and inverse trig functions
Logarithmic	log, ln, 10^x, exp	Common and natural logarithms, exponentials
Power Functions	x², x³, √, ³ √, ^	Square, cube, roots, general power
Constants	π, e	Mathematical constants
Advanced	n!, nCr, nPr	Factorial, combinations, permutations
Numeric	0-9, .	Number input and decimal point
Basic Operations	+, -, *, /	Arithmetic operations
Grouping	(,)	Expression grouping
Execution	=	Calculate result

Project Configuration

This document covers the project configuration, dependency management, build system setup, and development environment configuration for the scientific calculator application. It explains how project metadata is defined, how dependencies are managed using modern Python tooling, and how the build process is configured.

Project Metadata

The scientific calculator project is configured using a modern **pyproject.toml** file that defines the project metadata and dependencies in a standardized format.

Basic Project Information

The project is defined with the following metadata in **project.toml**:

Field	Value	Description
name	"scientific_calculator"	Package name for distribution
version	"0.1.0"	Current version following semantic versioning
description	"A modern scientific calculator with GUI"	Brief project description
requires-python	">=3.10"	Minimum Python version requirement

Dependency Management

The project uses a dual approach to dependency management, combining specification in **pyproject.toml** with precise version locking via **uv.lock**.

Core Dependencies

The application has two primary runtime dependencies defined in **project.toml**:

- **CustomTkinter (>=5.2.0)**: Modern GUI framework providing enhanced Tkinter widgets with contemporary styling
- **NumPy (>=1.24.0)**: Scientific computing library for advanced mathematical operations

Acknowledgement

We express our heartfelt gratitude to all the teachers for their invaluable guidance, support, and mentorship throughout the development of this project. Their expert advice, constructive feedback, and unwavering encouragement played a pivotal role in shaping this work from concept to completion.

We are also deeply thankful to the faculty and staff of the Department **of Computer Science and Technology** at **Bengal Institute of Technology** for providing the necessary resources, conducive environment, and academic inspiration that enabled us to successfully execute this major project.

We extend our sincere appreciation to our friends and family, whose constant motivation and emotional support sustained us throughout the journey.

Additionally, we acknowledge the creators and maintainers of the open-source tools and APIs that powered this project — including Spotify Web API, Google Gemini AI, React, Django, and Recoil. Their robust and accessible platforms made it possible to build a scalable, intelligent, and user-friendly application.

Finally, we thank our peers and supporters whose belief in our vision helped us stay disciplined, focused, and driven from start to finish.

Index Table

Sl. No.	Title	Page No.
1	Application Overview	1
2	System Requirements	2
3	Getting Started	3
4	Application Startup Sequence	4
5	System Architecture	5
6	Key Architectural Principles	6
7	Core Components	7
8	Core Component Mapping	8
9	Calculator Engine	10
10	GUI Application	12
11	User Interface Layout	13
12	User Interface Components	15
13	Project Configuration	16
14	Acknowledgement	17