CustomTkinter Calculator Project

1. Introduction

This document provides an overview of the CustomTkinter-based Calculator application. It explains the user interface, features, underlying logic, and necessary steps to run the project.

2. Features and Functionality

• Modern Dark/Light themes powered by CustomTkinter.
• Dynamic display textbox that adapts font size to expression length.
• History panel that shows past calculations and can be cleared.
• Percentage (%) button with custom logic (e.g., 20 - 60% yields 8).
• Square root (√) function with error handling for negative inputs.
• ANS button to reuse the last calculated result.
• Keyboard shortcuts for faster input (Enter, Backspace, r, t, etc.).
• Automatic prevention of invalid operator sequences (e.g., "++").

3. User Interface Overview

The main window is 350×650 px by default (minimum: 350×650, maximum: 400×750). It contains a read-only display textbox, an optional history panel, three control buttons at the top (History, Clear, Theme), and a 4×6 grid of calculator buttons for numbers and operators.

4. Keyboard Shortcuts

Key	Function
0-9, +, -, *, /, (,)	Enter numbers and operators
Enter	Evaluate the current expression
Backspace	Delete the last character
Escape	Clear the entire expression
r	Calculate square root (√)
t	Toggle Dark/Light theme
%	Calculate percentage
^	Power operation

5. Calculation Logic

Expressions are sanitized and transformed before evaluation. Custom symbols are mapped to Python equivalents (e.g., '÷' → '/', '×' → '*', '^' →

'**'). The eval() function computes the result, which is rounded to two decimal places where appropriate.

Percentage calculations apply contextual rules: for example, in ‘20 - 60%’, 60 % of 20 is subtracted, producing 8.

6. Technical Stack

• Python 3.x
• CustomTkinter (modern wrapper around Tkinter)
• Tkinter for underlying GUI components
• Object-Oriented Programming (single CalculatorApp class)
• Exception handling for division by zero and invalid input

7. Error Handling

The application gracefully handles common errors such as division by zero, invalid percentage syntax, and attempts to take the square root of negative numbers. User-friendly messages are displayed instead of program crashes.

8. How to Run the Application

1. Install the CustomTkinter library:
   pip install customtkinter

2. Ensure the icon file path (calcu.ico) is valid, or adjust the path in the code.

3. Run the script using Python:
   python calculator.py

9. Conclusion

This project demonstrates how CustomTkinter can be employed to build a modern, user-friendly calculator with advanced features such as theme switching, history tracking, and extended mathematical functions. It serves as an effective showcase of GUI development and event handling in Python.

Complete Guide to Building a GUI Calculator App in Python with customtkinter

This guide will help you understand and build a GUI-based calculator app for PC using Python and the customtkinter library.
It covers all the programming and UI concepts you need to learn step-by-step, explains the full example code, and provides a learning checklist.

Step-by-Step Learning Roadmap

1. Python Fundamentals:
- Variables, data types, functions, control flow, exception handling, OOP

2. Tkinter Basics:
- Creating windows, buttons, labels
- Using pack and grid for layouts
- Handling events and StringVar

3. CustomTkinter Library:
- Installing and using customtkinter
- Working with CTk widgets and themes

4. Expression Handling:
- Building calculator expressions as strings
- Replacing custom symbols (×, ÷, ^) for evaluation
- Using eval() safely with error handling

5. Event Binding:
- Keyboard input bindings
- Button commands

6. App Structure:
- Organizing code into a class
- Maintaining state and UI updates

7. Advanced Features:
- History view
- Theme toggle (light/dark mode)
- Special functions (square root, percentage)

Code Explanation and Guide

Your calculator app uses a class CalculatorApp inherited from ctk.CTk.

Key components:

- __init__: Initializes window, variables, and calls UI setup
- create_widgets: Builds main UI parts including display, buttons, history box
- build_buttons: Dynamically creates all calculator buttons with colors and commands
- press_key: Updates the expression string when user inputs
- evaluate: Calculates the result using Python's eval after replacing symbols
- do_square_root, do_percentage: Special operations
- toggle_history_view: Shows/hides the history panel
- flip_theme: Switches between light and dark mode and rebuilds buttons
- clear_all, delete_last: Editing the current expression
- handle_keypress: Maps keyboard keys to calculator functions

The buttons grid is created using grid geometry manager for a responsive layout.
The app maintains state with self.expression and self.history_shown.
History of calculations is stored in a CTkTextbox.

What to Learn & Practice

- Python basics and OOP: variables, functions, exceptions, classes
- Tkinter fundamentals: windows, buttons, labels, layout managers
- CustomTkinter: widgets, theming, appearance modes
- String manipulation and safe evaluation of expressions
- Event handling: button commands and keyboard bindings
- GUI app structure: class-based design, state management
- Handling UI updates dynamically
- Debugging and error handling
- Reading and understanding existing code

Practice building small projects incrementally:

1. Console calculator
2. Basic Tkinter GUI with buttons and labels
3. CustomTkinter themed window

4. Calculator with buttons and evaluation
5. Add keyboard support
6. Add history and theme toggle
7. Organize code into a class

Resources & Tips

- Automate the Boring Stuff with Python (free book)
- Real Python tutorials on OOP and Tkinter
- CustomTkinter GitHub repo and docs
- Stack Overflow for Q&A
- YouTube tutorials on Python GUI (freeCodeCamp, Tech With Tim)
- Practice small coding exercises regularly

Tip: Start small, test often, and gradually add features.
Use version control like Git to track your progress.

Python Code:

```python
import customtkinter as ctk
import tkinter.messagebox as mbox
import re
import os

ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("dark-blue")


class CalculatorApp(ctk.CTk):
    def __init__(self):
        super().__init__()


        icon_path = r"C:\Users\Salman\Documents\5th Semester Project\Me
        Practice\Python\calcu.ico"
        if os.path.exists(icon_path):
            self.iconbitmap(icon_path)
        else:
            print("Icon file not found:", icon_path)

        self.title("Calculator")
        self.geometry("350x650")
        self.maxsize(400, 750)
        self.minsize(350, 650)

        self.expression = ""
        self.last_result = ""
        self.is_dark_mode = True
        self.history_shown = False
        self.max_input_length = 100

        self.create_widgets()
        self.bind("<Key>", self.handle_keypress)
```

```
            ('(', lambda: self.press_key('(')), (')', lambda: self.press_key(')')), ('%', self.do_percentage), ('÷', lambda: self.press_key('÷')),
            ('7', lambda: self.press_key('7')), ('8', lambda: self.press_key('8')), ('9', lambda: self.press_key('9')), ('×', lambda: self.press_key('×')),
            ('4', lambda: self.press_key('4')), ('5', lambda: self.press_key('5')), ('6', lambda: self.press_key('6')), ('-', lambda: self.press_key('-')),
            ('1', lambda: self.press_key('1')), ('2', lambda: self.press_key('2')), ('3', lambda: self.press_key('3')), ('+', lambda: self.press_key('+')),
            ('ANS', self.insert_ans), ('0', lambda: self.press_key('0')), ('.', lambda: self.press_key('.')), ('=', self.evaluate),
        ]

        for idx, (txt, cmd) in enumerate(buttons):
            colors = get_color_settings(txt)
            btn = ctk.CTkButton(self.frame, text=txt, command=cmd, **self.btn_style, **colors)
            btn.grid(row=idx // 4, column=idx % 4, padx=5, pady=5, sticky="nsew")

        for i in range(4):
            self.frame.grid_columnconfigure(i, weight=1)
        for i in range(6):
            self.frame.grid_rowconfigure(i, weight=1)

        for btn in [self.history_btn, self.clear_history_btn, self.theme_btn]:
            colors = get_color_settings(btn.cget("text"))
            btn.configure(**colors)

    def get_display_font_size(self):
        length = len(self.expression)
        if length <= 10:
```

```
    def create_widgets(self):
        self.display_box = ctk.CTkTextbox(self, height=100, font=('Segoe UI', 16), activate_scrollbars=False, wrap="none")
        self.display_box.insert("1.0", "")
        self.display_box.configure(state="disabled")
        self.display_box.pack(fill='x', padx=10, pady=(10, 5))
        self.display_box.bind("<Button-1>", self.copy_result_to_clipboard)

        self.history_frame = ctk.CTkFrame(self, fg_color="transparent")
        self.history_box = ctk.CTkTextbox(self.history_frame, height=90, font=('Consolas', 20), fg_color="transparent",
                                          text_color="#999999",
                                          activate_scrollbars=False,
                                          wrap="none")
        self.history_box.pack(fill='both', expand=True)
        self.history_box.configure(state="disabled")

        self.top_row = ctk.CTkFrame(self, fg_color="transparent")
        self.top_row.pack(fill='x', pady=(5, 5))

        self.btn_style = {'font': ('Segoe UI', 18), 'height': 40, 'corner_radius': 10, 'width': 80}

        self.history_btn = ctk.CTkButton(self.top_row, text="History", command=self.toggle_history_view, **self.btn_style)
        self.history_btn.pack(side='left', padx=(10, 5))

        self.clear_history_btn = ctk.CTkButton(self.top_row, text="Clear", command=self.clear_history, **self.btn_style)
        self.clear_history_btn.pack_forget()

        self.theme_btn = ctk.CTkButton(self.top_row, text="🌞 Light", command=self.flip_theme, **self.btn_style)
        self.theme_btn.pack(side='right', padx=(5, 10))
```

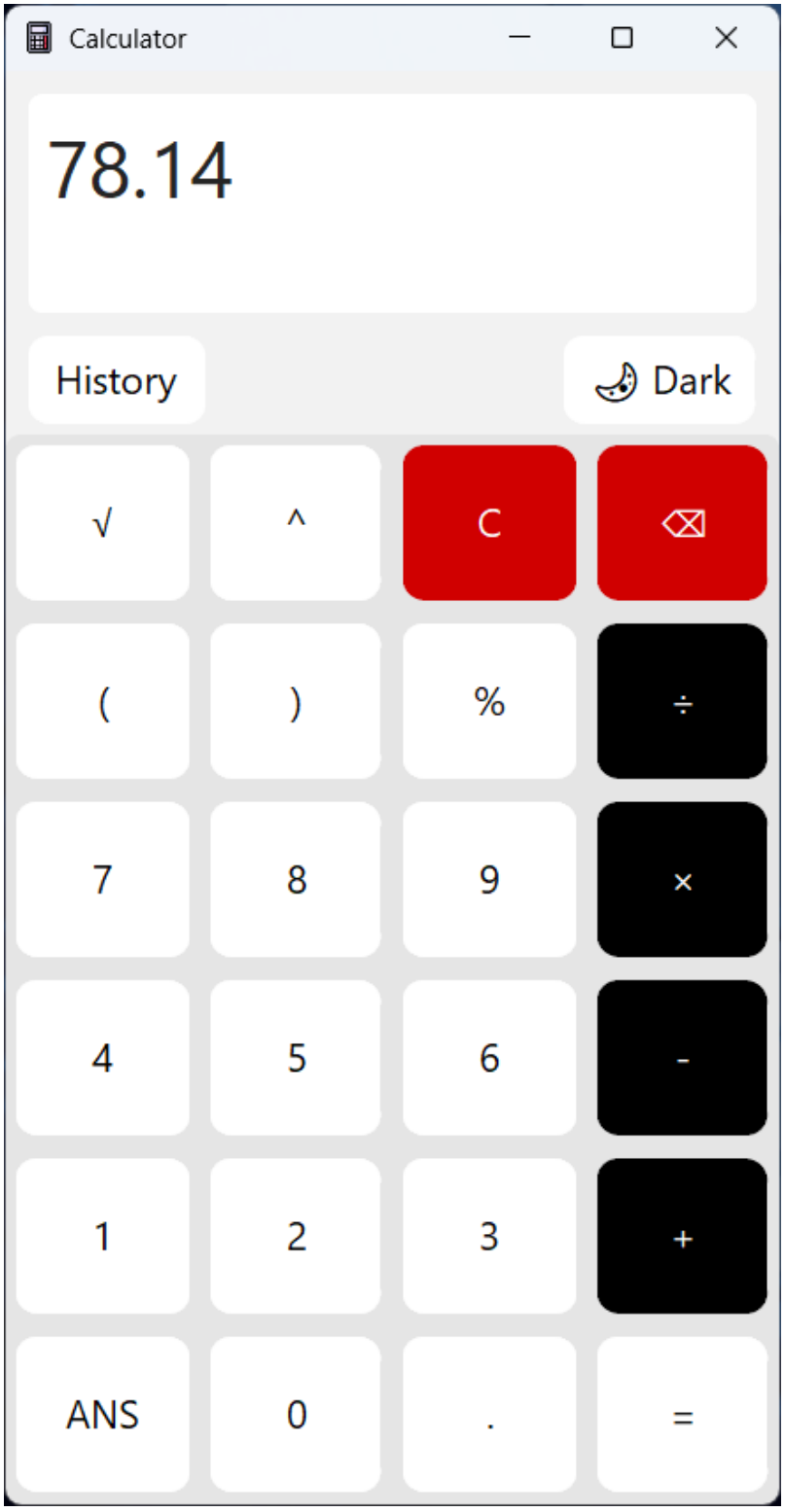
```
        self.frame = ctk.CTkFrame(self)
        self.frame.pack(fill='both', expand=True)

        self.build_buttons()

    def build_buttons(self):
        self.frame.destroy()
        self.frame = ctk.CTkFrame(self)
        self.frame.pack(fill='both', expand=True)

        def get_color_settings(text):
            white = "#ffffff"
            black = "#000000"
            red = "#d00000"
            dark_bg = "#2c2c2c"
            hover = "#dcdcdc" if not self.is_dark_mode else "#3a3a3a"

            if text in ('C', '⌫'):
                return {'fg_color': red, 'text_color': white, 'hover_color':
                hover}
            elif text in ('+', '-', '×', '÷'):
                return {'fg_color': black, 'text_color': white, 'hover_color':
                hover}
            else:
                return {
                    'fg_color': white if not self.is_dark_mode else dark_bg,
                    'text_color': black if not self.is_dark_mode else white,
                    'hover_color': hover
                }

        buttons = [
            ('√', self.do_square_root), ('^', lambda: self.press_key('^')),
            ('C', self.clear_all), ('⌫', self.delete_last),
            ('(', lambda: self.press_key('(')), (')', lambda: self.press_key(')
            return ('Segoe UI', 36)
        elif length <= 20:
            return ('Segoe UI', 28)
        elif length <= 30:
            return ('Segoe UI', 22)
        else:
            return ('Segoe UI', 16)

    def update_display(self):
        self.display_box.configure(state="normal")
        self.display_box.delete("1.0", "end")
        self.display_box.insert("1.0", self.expression)
        self.display_box.configure(font=self.get_display_font_size())
        self.display_box.configure(state="disabled")

    def press_key(self, val):
        if len(self.expression) >= self.max_input_length:
            mbox.showwarning("Limit", "Maximum input length reached!")
            return

        operators = "+-×÷^"
        if val in operators:
            if not self.expression:
                return
            if self.expression[-1] in operators:
                self.expression = self.expression[:-1]
            self.expression += val
            self.update_display()
            return

        self.expression += val
        self.update_display()

    def evaluate(self):
```

```
        try:
            expr = self.expression.replace('÷', '/').replace('×', '*').replace
            ('^', '**')
            if self.last_result:
                expr = expr.replace('ANS', self.last_result)

            expr = re.sub(r'\b0+(\d)', r'\1', expr)

            result = eval(expr)
            self.last_result = str(result)
            rounded = round(result, 2)
            result_display = int(rounded) if rounded == int(rounded) else
            rounded

            self.history_box.configure(state="normal")
            self.history_box.insert("end", f"{self.expression} =
            {result_display}\n")
            self.history_box.configure(state="disabled")

            self.expression = str(result_display)
            self.update_display()
        except ZeroDivisionError:
            self.expression = "Can't divide by zero"
            self.update_display()
            self.expression = ""
        except Exception:
            self.expression = "Error"
            self.update_display()
            self.expression = ""

    def clear_all(self):
        self.expression = ""
        self.update_display()
def delete_last(self):
    self.expression = self.expression[:-1]
    self.update_display()

def do_square_root(self):
    try:
        if not self.expression:
            return
        expr = self.expression.replace('÷', '/').replace('×', '*').replace
        ('^', '**')
        if self.last_result:
            expr = expr.replace('ANS', self.last_result)
        val = eval(expr)
        if val < 0:
            raise ValueError("Invalid Input")

        sqrt_val = val ** 0.5
        rounded = round(sqrt_val, 2)
        result_display = int(rounded) if rounded == int(rounded) else
        rounded
        self.last_result = str(result_display)
        self.expression = self.last_result

        self.update_display()

        self.history_box.configure(state="normal")
        self.history_box.insert("end", f"√({val}) = {result_display}\n")
        self.history_box.configure(state="disabled")

    except:
        self.expression = "Error"
        self.update_display()
        self.expression = ""

```

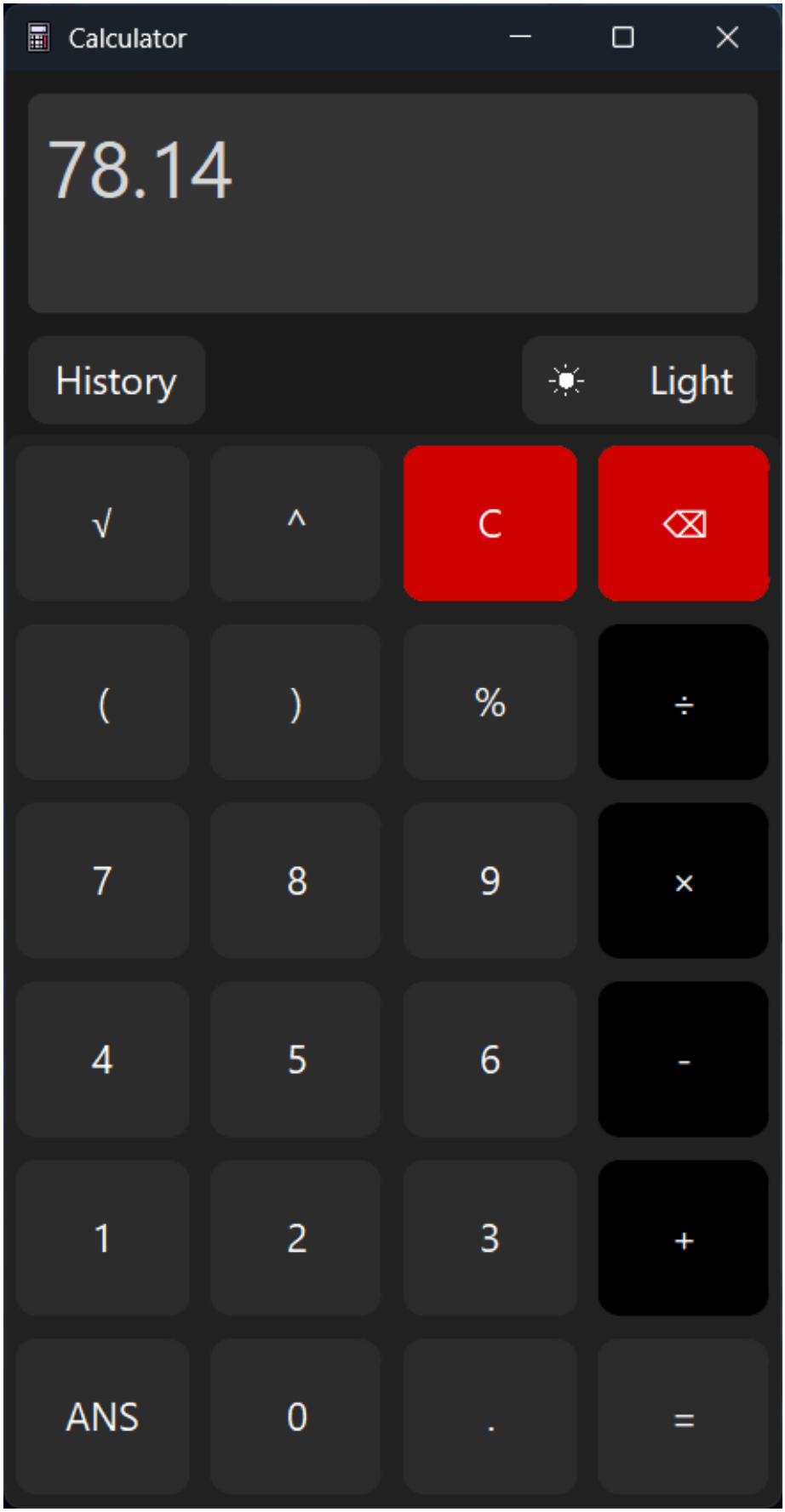
```
212     def do_percentage(self):
213         try:
214             if not self.expression:
215                 return
216             expr = self.expression.replace('×', '*').replace('÷', '/').replace
                ('^', '**')
217             if self.last_result:
218                 expr = expr.replace('ANS', self.last_result)
219             original = self.expression + '%'
220             for op in "+-*/":
221                 if op in expr:
222                     parts = expr.rsplit(op, 1)
223                     if len(parts) == 2:
224                         left = float(parts[0])
225                         right = float(parts[1])
226                         res = {
227                             '+': left + (left * right / 100),
228                             '-': left - (left * right / 100),
229                             '*': left * (right / 100),
230                             '/': left / (right / 100) if right != 0 else float
                            ('inf')
231                         }[op]
232                         break
233             else:
234                 res = float(expr) / 100
235             self.last_result = str(res)
236             final = round(res, 2)
237             final = int(final) if final == int(final) else final
238             self.expression = str(final)
239             self.update_display()
240             self.history_box.configure(state="normal")
241             self.history_box.insert("end", f"{original} = {final}\n")
242             self.history_box.configure(state="disabled")
243         except:
243         except:
244             self.expression = "Error"
245             self.update_display()
246             self.expression = ""
247 
248     def toggle_history_view(self):
249         self.history_shown = not self.history_shown
250         if self.history_shown:
251             self.history_frame.pack(after=self.display_box, fill='both',
                expand=False, padx=10, pady=(5, 5))
252             self.clear_history_btn.pack(side='left', padx=5)
253         else:
254             self.history_frame.pack_forget()
255             self.clear_history_btn.pack_forget()
256 
257     def flip_theme(self):
258         self.is_dark_mode = not self.is_dark_mode
259         mode = "dark" if self.is_dark_mode else "light"
260         self.theme_btn.configure(text="🌞 Light" if mode == "dark" else "🌙
            Dark")
261         ctk.set_appearance_mode(mode)
262 
263         if self.history_shown:
264             self.history_frame.pack_forget()
265             self.clear_history_btn.pack_forget()
266 
267         self.build_buttons()
268 
269         if self.history_shown:
270             self.history_frame.pack(after=self.display_box, fill='both',
                expand=False, padx=10, pady=(5, 5))
271             self.clear_history_btn.pack(side='left', padx=5)
272 
273     def clear_history(self):
```

```
    def clear_history(self):
        self.history_box.configure(state="normal")
        self.history_box.delete("1.0", "end")
        self.history_box.configure(state="disabled")

    def insert_ans(self):
        if self.last_result:
            self.press_key(self.last_result)

    def handle_keypress(self, evt):
        key = evt.char
        sym = evt.keysym

        if key in '0123456789.+-*/()':
            self.press_key(key.replace('*', '×').replace('/', '÷'))
        elif key.lower() == 'a':
            self.press_key("ANS")
        elif key == '\r':
            self.evaluate()
        elif sym == 'BackSpace':
            self.delete_last()
        elif sym == 'Escape':
            self.clear_all()
        elif key == '^':
            self.press_key('^')
        elif key == '%':
            self.do_percentage()
        elif key.lower() == 'r':
            self.do_square_root()
        elif sym.lower() == 't':
            self.flip_theme()

    def copy_result_to_clipboard(self, event):
        self.clipboard_clear()
```

```
        key = evt.char
        sym = evt.keysym

        if key in '0123456789.+-*/()':
            self.press_key(key.replace('*', '×').replace('/', '÷'))
        elif key.lower() == 'a':
            self.press_key("ANS")
        elif key == '\r':
            self.evaluate()
        elif sym == 'BackSpace':
            self.delete_last()
        elif sym == 'Escape':
            self.clear_all()
        elif key == '^':
            self.press_key('^')
        elif key == '%':
            self.do_percentage()
        elif key.lower() == 'r':
            self.do_square_root()
        elif sym.lower() == 't':
            self.flip_theme()

    def copy_result_to_clipboard(self, event):
        self.clipboard_clear()
        self.clipboard_append(self.expression)
        self.update()


if __name__ == "__main__":
    app = CalculatorApp()
    app.mainloop()
```

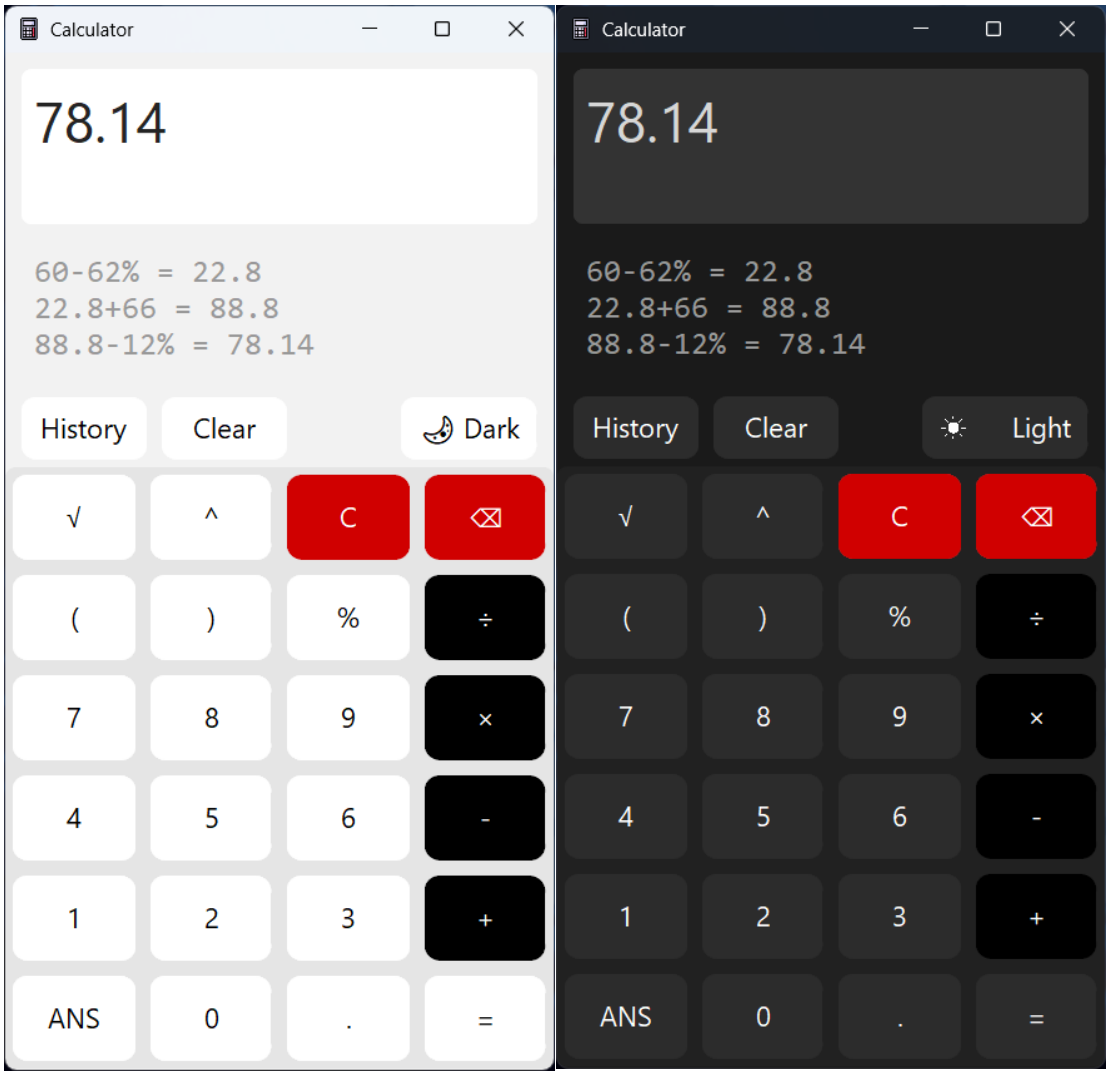


Output: 1



Output: 2

Output: 3


Calculator
78.14
60-62% = 22.8
22.8+66 = 88.8
88.8-12% = 78.14
History
Clear
Dark
√
^
C
⌫
(
)
%
÷
7
8
9
×
4
5
6
-
1
2
3
+
ANS
0
.
=
Calculator
78.14
60-62% = 22.8
22.8+66 = 88.8
88.8-12% = 78.14
History
Clear
Light
√
^
C
⌫
(
)
%
÷
7
8
9
×
4
5
6
-
1
2
3
+
ANS
0
.
=