

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 ($\sqrt{}$) 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 ($\sqrt{}$)
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 (\times , \div , $^$) 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:

```
1  import customtkinter as ctk
2  import tkinter.messagebox as mbox
3  import re
4  import os
5
6  ctk.set_appearance_mode("dark")
7  ctk.set_default_color_theme("dark-blue")
8
9
10 class CalculatorApp(ctk.CTk):
11     def __init__(self):
12         super().__init__()
13
14
15         icon_path = r"C:\Users\Salman\Documents\5th Semester Project\Me
Practice\Python\calcu.ico"
16         if os.path.exists(icon_path):
17             self.iconbitmap(icon_path)
18         else:
19             print("Icon file not found:", icon_path)
20
21         self.title("Calculator")
22         self.geometry("350x650")
23         self.maxsize(400, 750)
24         self.minsize(350, 650)
25
26         self.expression = ""
27         self.last_result = ""
28         self.is_dark_mode = True
29         self.history_shown = False
30         self.max_input_length = 100
31
32         self.create_widgets()
33         self.bind("<Key>", self.handle_keypress)
```

```

91         ('(', lambda: self.press_key('(')), (')', lambda: self.press_key(')'),
92         ('%', self.do_percentage), ('÷', lambda: self.press_key('÷')),
93         ('7', lambda: self.press_key('7')), ('8', lambda: self.press_key('8')),
94         ('9', lambda: self.press_key('9')), ('×', lambda: self.press_key('×')),
95         ('4', lambda: self.press_key('4')), ('5', lambda: self.press_key('5')),
96         ('6', lambda: self.press_key('6')), ('-', lambda: self.press_key('-')),
97         ('1', lambda: self.press_key('1')), ('2', lambda: self.press_key('2')),
98         ('3', lambda: self.press_key('3')), ('+', lambda: self.press_key('+')),
99         ('ANS', self.insert_ans), ('0', lambda: self.press_key('0')), ('.', lambda: self.press_key('.')),
100         ('=', self.evaluate),
101     ]
102
103     for idx, (txt, cmd) in enumerate(buttons):
104         colors = get_color_settings(txt)
105         btn = ctk.CTkButton(self.frame, text=txt, command=cmd, **self.btn_style, **colors)
106         btn.grid(row=idx // 4, column=idx % 4, padx=5, pady=5, sticky="nsew")
107
108     for i in range(4):
109         self.frame.grid_columnconfigure(i, weight=1)
110     for i in range(6):
111         self.frame.grid_rowconfigure(i, weight=1)
112
113     for btn in [self.history_btn, self.clear_history_btn, self.theme_btn]:
114         colors = get_color_settings(btn.cget("text"))
115         btn.configure(**colors)
116
117 def get_display_font_size(self):
118     length = len(self.expression)
119     if length <= 10:
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

61     self.frame = ctk.CTkFrame(self)
62     self.frame.pack(fill='both', expand=True)
63
64     self.build_buttons()
65
66     def build_buttons(self):
67         self.frame.destroy()
68         self.frame = ctk.CTkFrame(self)
69         self.frame.pack(fill='both', expand=True)
70
71     def get_color_settings(text):
72         white = "#ffffff"
73         black = "#000000"
74         red = "#d00000"
75         dark_bg = "#2c2c2c"
76         hover = "#dcdcdc" if not self.is_dark_mode else "#3a3a3a"
77
78         if text in ('C', '⊗'):
79             return {'fg_color': red, 'text_color': white, 'hover_color':
80                     hover}
81         elif text in ('+', '-', '⊗', '÷'):
82             return {'fg_color': black, 'text_color': white, 'hover_color':
83                     hover}
84         else:
85             return {
86                 'fg_color': white if not self.is_dark_mode else dark_bg,
87                 'text_color': black if not self.is_dark_mode else white,
88                 'hover_color': hover
89             }
90
91     buttons = [
92         ('V', self.do_square_root), ('^', lambda: self.press_key('^')),
93         ('C', self.clear_all), ('⊗', self.delete_last),
94         ('(', lambda: self.press_key('(')), (')', lambda: self.press_key(')'))
95
115         return ('Segoe UI', 36)
116     elif length <= 20:
117         return ('Segoe UI', 28)
118     elif length <= 30:
119         return ('Segoe UI', 22)
120     else:
121         return ('Segoe UI', 16)
122
123     def update_display(self):
124         self.display_box.configure(state="normal")
125         self.display_box.delete("1.0", "end")
126         self.display_box.insert("1.0", self.expression)
127         self.display_box.configure(font=self.get_display_font_size())
128         self.display_box.configure(state="disabled")
129
130     def press_key(self, val):
131         if len(self.expression) >= self.max_input_length:
132             mbox.showwarning("Limit", "Maximum input length reached!")
133             return
134
135         operators = "+-x÷^"
136         if val in operators:
137             if not self.expression:
138                 return
139             if self.expression[-1] in operators:
140                 self.expression = self.expression[:-1]
141             self.expression += val
142             self.update_display()
143             return
144
145         self.expression += val
146         self.update_display()
147
148     def evaluate(self):

```



```

149         try:
150             expr = self.expression.replace('÷', '/').replace('x', '*').replace(
151                 ('^', '**'))
152             if self.last_result:
153                 expr = expr.replace('ANS', self.last_result)
154
155             expr = re.sub(r'\b0+(\d)', r'\1', expr)
156
157             result = eval(expr)
158             self.last_result = str(result)
159             rounded = round(result, 2)
160             result_display = int(rounded) if rounded == int(rounded) else
161                 rounded
162
163             self.history_box.configure(state="normal")
164             self.history_box.insert("end", f"{self.expression} =
165                 {result_display}\n")
166             self.history_box.configure(state="disabled")
167
168             self.expression = str(result_display)
169             self.update_display()
170         except ZeroDivisionError:
171             self.expression = "Can't divide by zero"
172             self.update_display()
173             self.expression = ""
174         except Exception:
175             self.expression = "Error"
176             self.update_display()
177             self.expression = ""
178
179     def clear_all(self):
180         self.expression = ""
181         self.update_display()
182
183     def delete_last(self):
184         self.expression = self.expression[:-1]
185         self.update_display()
186
187     def do_square_root(self):
188         try:
189             if not self.expression:
190                 return
191             expr = self.expression.replace('÷', '/').replace('x', '*').replace(
192                 ('^', '**'))
193             if self.last_result:
194                 expr = expr.replace('ANS', self.last_result)
195             val = eval(expr)
196             if val < 0:
197                 raise ValueError("Invalid Input")
198
199             sqrt_val = val ** 0.5
200             rounded = round(sqrt_val, 2)
201             result_display = int(rounded) if rounded == int(rounded) else
202                 rounded
203             self.last_result = str(result_display)
204             self.expression = self.last_result
205
206             self.update_display()
207
208             self.history_box.configure(state="normal")
209             self.history_box.insert("end", f"√({val}) = {result_display}\n")
210             self.history_box.configure(state="disabled")
211
212         except:
213             self.expression = "Error"
214             self.update_display()
215             self.expression = ""

```

```

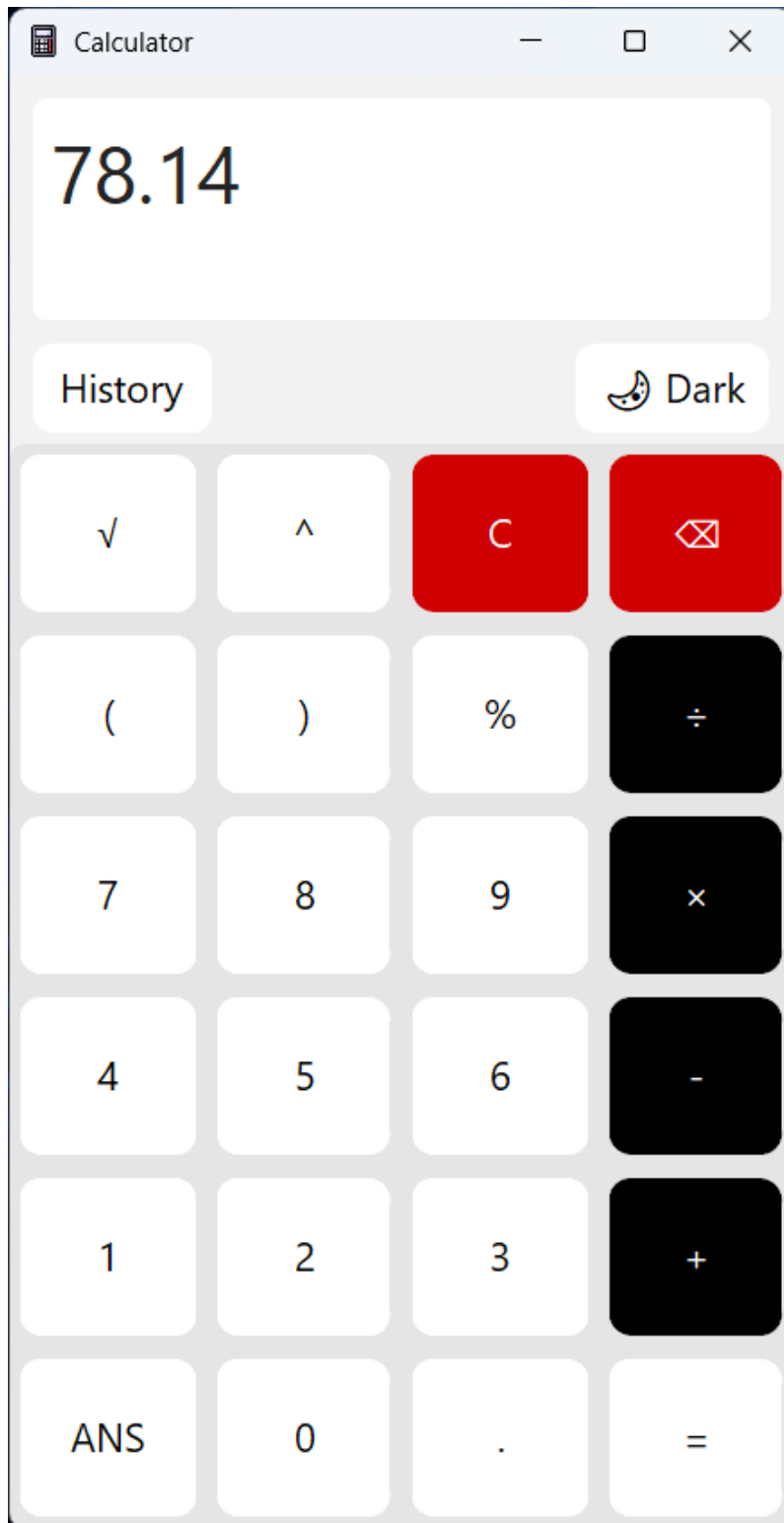
212 def do_percentage(self):
213     try:
214         if not self.expression:
215             return
216         expr = self.expression.replace('x', '*').replace('÷', '/').replace(
217             '^', '**')
218         if self.last_result:
219             expr = expr.replace('ANS', self.last_result)
220         original = self.expression + '%'
221         for op in "+-*/":
222             if op in expr:
223                 parts = expr.rsplit(op, 1)
224                 if len(parts) == 2:
225                     left = float(parts[0])
226                     right = float(parts[1])
227                     res = {
228                         '+': left + (left * right / 100),
229                         '-': left - (left * right / 100),
230                         '*': left * (right / 100),
231                         '/': left / (right / 100) if right != 0 else float(
232                             'inf')
233                     }[op]
234                     break
235             else:
236                 res = float(expr) / 100
237                 self.last_result = str(res)
238                 final = round(res, 2)
239                 final = int(final) if final == int(final) else final
240                 self.expression = str(final)
241                 self.update_display()
242                 self.history_box.configure(state="normal")
243                 self.history_box.insert("end", f"{original} = {final}\n")
244                 self.history_box.configure(state="disabled")
245     except:
246         self.expression = "Error"
247         self.update_display()
248         self.expression = ""
249
250 def toggle_history_view(self):
251     self.history_shown = not self.history_shown
252     if self.history_shown:
253         self.history_frame.pack(after=self.display_box, fill='both',
254             expand=False, padx=10, pady=(5, 5))
255         self.clear_history_btn.pack(side='left', padx=5)
256     else:
257         self.history_frame.pack_forget()
258         self.clear_history_btn.pack_forget()
259
260 def flip_theme(self):
261     self.is_dark_mode = not self.is_dark_mode
262     mode = "dark" if self.is_dark_mode else "light"
263     self.theme_btn.configure(text="☀ Light" if mode == "dark" else "🌙
264     Dark")
265     ctk.set_appearance_mode(mode)
266
267 if self.history_shown:
268     self.history_frame.pack_forget()
269     self.clear_history_btn.pack_forget()
270
271 self.build_buttons()
272
273 if self.history_shown:
274     self.history_frame.pack(after=self.display_box, fill='both',
275         expand=False, padx=10, pady=(5, 5))
276     self.clear_history_btn.pack(side='left', padx=5)
277
278 def clear_history(self):

```

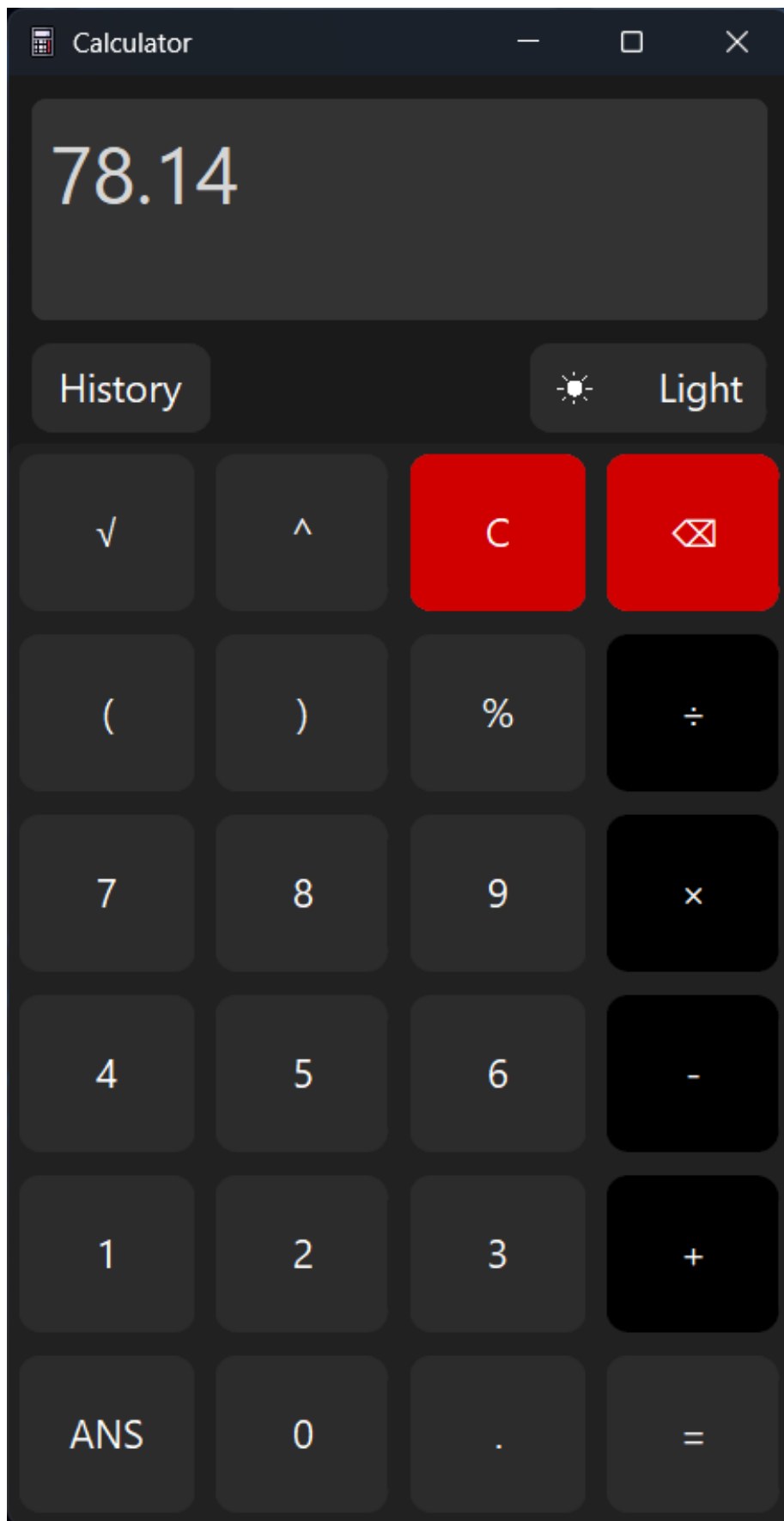
```

273     def clear_history(self):
274         self.history_box.configure(state="normal")
275         self.history_box.delete("1.0", "end")
276         self.history_box.configure(state="disabled")
277
278     def insert_ans(self):
279         if self.last_result:
280             self.press_key(self.last_result)
281
282     def handle_keypress(self, evt):
283         key = evt.char
284         sym = evt.keysym
285
286         if key in '0123456789.+*/()':
287             self.press_key(key.replace('*', '⌘').replace('/', '÷'))
288         elif key.lower() == 'a':
289             self.press_key("ANS")
290         elif key == '\r':
291             self.evaluate()
292         elif sym == 'BackSpace':
293             self.delete_last()
294         elif sym == 'Escape':
295             self.clear_all()
296         elif key == '^':
297             self.press_key('^')
298         elif key == '%':
299             self.do_percentage()
300         elif key.lower() == 'r':
301             self.do_square_root()
302         elif sym.lower() == 't':
303             self.flip_theme()
304
305     def copy_result_to_clipboard(self, event):
306         self.clipboard_clear()
307
308         key = evt.char
309         sym = evt.keysym
310
311         if key in '0123456789.+*/()':
312             self.press_key(key.replace('*', '⌘').replace('/', '÷'))
313         elif key.lower() == 'a':
314             self.press_key("ANS")
315         elif key == '\r':
316             self.evaluate()
317         elif sym == 'BackSpace':
318             self.delete_last()
319         elif sym == 'Escape':
320             self.clear_all()
321         elif key == '^':
322             self.press_key('^')
323         elif key == '%':
324             self.do_percentage()
325         elif key.lower() == 'r':
326             self.do_square_root()
327         elif sym.lower() == 't':
328             self.flip_theme()
329
330     def copy_result_to_clipboard(self, event):
331         self.clipboard_clear()
332         self.clipboard_append(self.expression)
333         self.update()
334
335 if __name__ == "__main__":
336     app = CalculatorApp()
337     app.mainloop()

```



Output: 1



Output: 2

Output: 3

