Report On

Scientific Calculator

Submitted in partial fulfillment of the requirements of the Course project in Semester IV of Second Year Computer Engineering

by
Sanika Gharat (Roll No. 76)
Jidnyasa Naik (Roll No. 79)

Supervisor
Prof. Sneha Mhatre

Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

(2024-25)

Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

CERTIFICATE

This is to certify that the project entitled “Scientific Calculator” is a bonafide work of "Sanika Gharat (Roll No. 76), Jidnyasa Naik (Roll No. 79)" submitted to the University of Mumbai in partial fulfillment of the requirement for the Course project in semester IV of Second Year Computer Engineering.

Supervisor

Prof. Sneha Mhatre

Dr Megha Trivedi
Head of Department

Dr. H.V. Vankudre
Principal

Abstract

This project aims to design and implement a Graphical User Interface (GUI) for a Scientific Calculator. The traditional calculator experience is transformed into a user-friendly interface, providing convenience and enhanced interaction for users. The project utilizes Python programming language and relevant libraries such as Tkinter for GUI development The project involves several stages of development, including conceptualization, design, implementation, testing, and documentation. Throughout the process, attention is paid to user experience and interface design principles to create an engaging and seamless interaction environment. By the completion of this project, a fully functional GUI-based Scientific Calculator will be delivered, catering to the needs of tabletop gaming enthusiasts and users requiring random number generation in various scenarios. The project not only serves as a practical application but also demonstrates the utilization of GUI development techniques using Python, contributing to the advancement of software development skills.

INDEX

Contents

INTRODUCTION

1.1 Introduction

This Python script implements a graphical calculator using Tkinter, enabling users to perform basic arithmetic operations conveniently. Notably, it features a history function that stores previous calculations for reference. Upon execution, the calculator interface displays an input field and buttons for digits, operators, and functional commands like clearing input and accessing history. Users can input expressions via mouse clicks or keyboard input. Pressing 'Enter' computes the expression, 'H' displays past calculations, and 'CH' clears the history. Behind the scenes, SQLite is utilized to securely store calculation history.

1.2 Problem Statement

The task at hand involves developing a Python script to construct a graphical calculator application equipped with basic arithmetic functionalities and an integrated history feature. Using the Tkinter library, the application's user interface will offer a straightforward layout, featuring an entry widget for input and buttons for digits, operators, and functional commands. Users will be able to interact with the calculator using both mouse clicks and keyboard input, ensuring flexibility and convenience. Furthermore, the calculator will incorporate a history function, storing previous calculations securely in an SQLite database for easy reference. Key requirements include accurate arithmetic operations, intuitive user interface design, seamless interaction via keyboard, reliable history logging, and the implementation of functional commands to manage input and access history logs. Upon completion, the Python script will deliver a fully functional calculator application with a history feature, meeting the specified criteria for functionality, usability, and reliability. Accompanying documentation will provide insights into the implemented functionalities, usage instructions, and considerations for future maintenance and enhancements.

PROPOSED SYSTEM

Key features of the proposed system include:

1. **Graphical User Interface (GUI):** The calculator interface is designed using Tkinter, ensuring a visually appealing and intuitive layout. It includes an entry widget for input and a set of buttons representing digits (0-9), arithmetic operators (+, -, *, /), and functional commands.

2. **Arithmetic Operations**: Users can perform basic arithmetic operations, such as addition, subtraction, multiplication, and division, by entering expressions either via mouse clicks or keyboard input**.**

3. **Keyboard Interaction:** The calculator supports keyboard interaction, allowing users to input expressions and execute commands using key bindings for increased efficiency and accessibility.

4. **History Feature:** One of the standout features of the calculator is its history function, which records and stores previous calculations securely in an SQLite database. Users can access their calculation history, review past expressions and results, and clear the history log as needed**.**

5. **Database Integration:** SQLite integration ensures seamless management and storage of calculation history, providing persistence across multiple sessions and enhancing the reliability of the application**.**

6. **Functional Commands:** The calculator includes functional commands to facilitate user interaction, such as clearing the current input, computing results, accessing the history log, and clearing the entire history**.**

2.2 Module Description:

Module Description: The Scientific Calculator Using Python module offers an introduction to building a scientific calculator application using the Python programming language. Participants will delve into fundamental concepts of Python programming, including variables, data types, functions, and control flow structures. Through step-by-step tutorials and hands-on exercises, participants will learn how to implement scientific calculator features such as arithmetic operations, trigonometric functions, logarithms, exponentiation, and statistical calculations using Python libraries such as math and numpy.

Additionally, participants will explore graphical user interface (GUI) development using libraries like Tkinter to create a user-friendly interface for the calculator application. By the end of the module, participants will have the skills to design and develop their own scientific calculator program in Python, enabling them to customize functionalities and expand their programming capabilities in scientific computing and data analysis tasks.

2.3 Description of Software & Hardware Used And Its Programming:

Software:

Python: The primary programming language used for development, providing a versatile and efficient platform for building the application.

Tkinter: A standard GUI library for Python, utilized for creating the graphical user interface of the Dice Roller application.

Integrated Development Environment (IDE): Software tools like PyCharm, Visual Studio Code, or IDLE are employed for coding, debugging, and testing the application code efficiently.

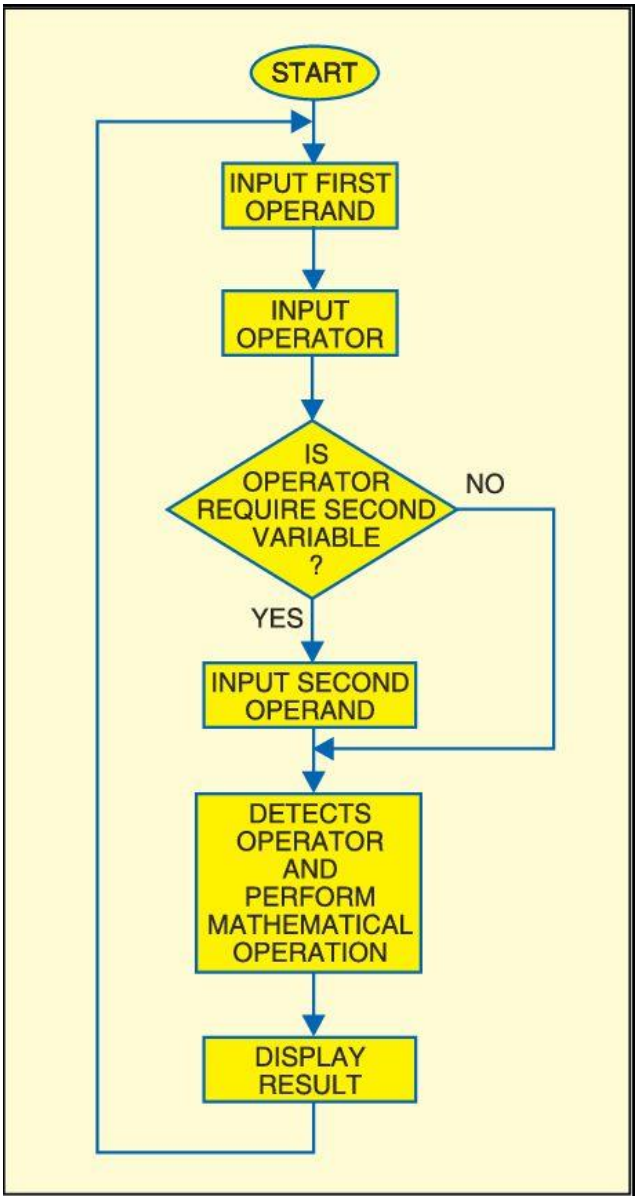
Operating System: The application is developed and tested on various operating systems such as Windows, macOS, and Linux to ensure cross-platform compatibility.

Hardware:

Personal Computers/Laptops: Developers utilized standard personal computers or laptops for coding and testing purposes.

Internet Connection: A stable internet connection was necessary for accessing online resources, libraries, and testing the web application across variousdevices and browsers.

Block Diagram:



2.4 Code:

```
from tkinter import *
import sqlite3

class Calculator:

    def _init_(self, master):
        # Assign reference to the main window of the application
        self.master = master
        # Add a name to our application
        master.title("Python Calculator")
        # Create a line where we display the equation
        self.equation = Entry(master, width=36, borderwidth=5)
        # Assign a position for the equation line in the grey application
window
        self.equation.grid(row=0, column=0, columnspan=4, padx=10,
pady=10)
        # Execute the .createButton() method
        self.createButton()
        # Create history text widget
        self.history_text = Text(master, height=10, width=20, bg="light
gray", fg="black", font=('Arial', 12))
        self.history_text.grid(row=6, column=0, columnspan=5, sticky=NSEW)
        # Create a connection to the SQLite database
        self.conn = sqlite3.connect('calhist.db')
        self.create_table()

        # Bind keyboard events
        for i in range(10):
            master.bind(str(i), lambda event, i=i:
self.clickButton(str(i)))
        master.bind('+', lambda event: self.clickButton('+'))
        master.bind('-', lambda event: self.clickButton('-'))
        master.bind('', lambda event: self.clickButton(''))
        master.bind('/', lambda event: self.clickButton('/'))
        master.bind('<Return>', lambda event: self.clickButton('='))
        master.bind('c', lambda event: self.clickButton('c'))
        master.bind('H', lambda event: self.clickButton('H'))
        master.bind('CH', lambda event: self.clickButton('CH'))

    def createButton(self):
        b0 = self.addButton(0)
        b1 = self.addButton(1)
        b2 = self.addButton(2)
        b3 = self.addButton(3)
        b4 = self.addButton(4)
        b5 = self.addButton(5)
```

```
        b6 = self.addButton(6)
        b7 = self.addButton(7)
        b8 = self.addButton(8)
        b9 =  self.addButton(9)
        b_add = self.addButton('+')
        b_sub = self.addButton('-')
        b_mult = self.addButton('*')
        b_div = self.addButton('/')
        b_clear = self.addButton('c')
        b_equal = self.addButton('=')
        hist = self.addButton("H")
        clrhist = self.addButton("CH")

        #Arrange the buttons into lists which represent calculator rows
        row1=[b7,b8,b9,b_add]
        row2=[b4,b5,b6,b_sub]
        row3=[b1,b2,b3,b_mult]
        row4=[b_clear,b0,b_equal,b_div]
        row5=[hist, clrhist]

        #Assign each button to a particular location on the GUI
        r=1
        for row in [row1, row2, row3, row4, row5]:
            c=0
            for buttn in row:
                buttn.grid(row=r, column=c, columnspan=1)
                c+=1
            r+=1

    def addButton(self,value):
        return Button(self.master, text=value, width=9, command = lambda:
self.clickButton(str(value)))

    def clickButton(self, value):
        # Get the equation that's entered by the user
        current_equation = str(self.equation.get())

        # If user clicked "c", then clear the screen
        if value == 'c':
            self.equation.delete(0, END)

        # If user clicked "=", then compute the answer and display it
        elif value == '=':
            try:
                answer = str(eval(current_equation))
                self.equation.delete(0, END)
                self.equation.insert(0, answer)
```

```
                self.save_to_history(current_equation + '=' + answer)  #
Save expression to history
            except Exception as e:
                self.equation.delete(0, END)
                self.equation.insert(0, 'Error')

        # If user clicked "H", show history
        elif value == 'H':
            self.show_history()

        # If user clicked "CH", clear history
        elif value == 'CH':
            self.clear_history()

        # If user clicked any other button, then add it to the equation
line
        else:
            self.equation.delete(0, END)
            self.equation.insert(0, current_equation + value)

    def create_table(self):
        cursor = self.conn.cursor()
        cursor.execute('''CREATE TABLE IF NOT EXISTS history (
                            id INTEGER PRIMARY KEY AUTOINCREMENT,
                            expression TEXT NOT NULL
                          )''')
        self.conn.commit()

    def clear_history(self):
        cursor = self.conn.cursor()
        cursor.execute("DELETE FROM history")
        self.conn.commit()
        self.history_text.config(state=NORMAL)  # Set state to normal to
allow modifications
        self.history_text.delete(1.0, END)  # Clear the text widget
        self.history_text.config(state=DISABLED)  # Set state back to
disabled to make it read-only

    def save_to_history(self, expression):
        cursor = self.conn.cursor()
        cursor.execute("INSERT INTO history (expression) VALUES (?)",
(expression,))
        self.conn.commit()

    def show_history(self):
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM history")
```

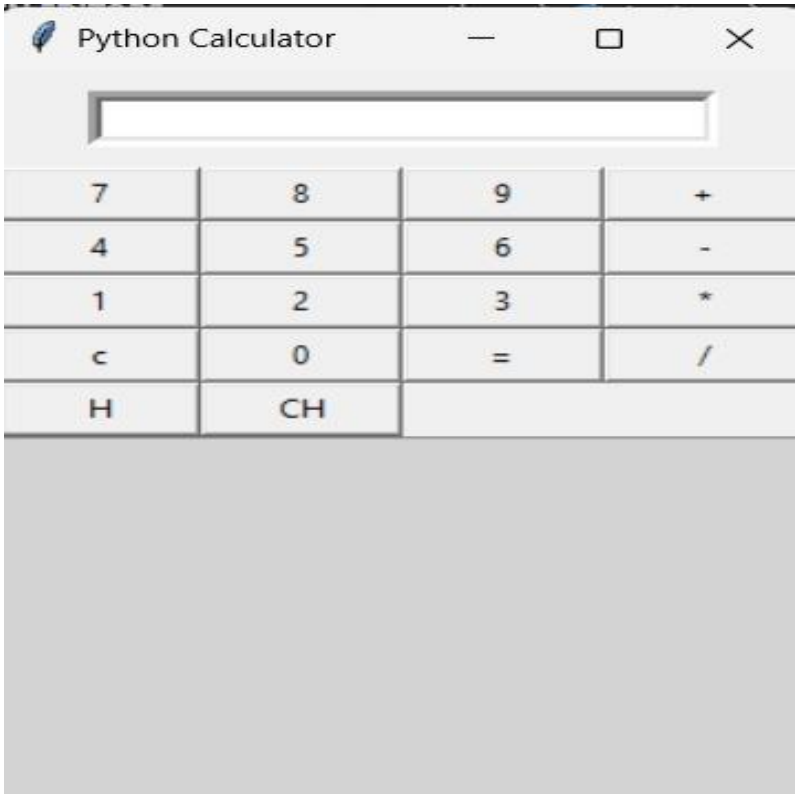
```
        history = cursor.fetchall()
        if history:
            self.history_text.config(state=NORMAL)  # Set state to normal
to allow modifications
            self.history_text.delete(1.0, END)  # Clear previous history
            for item in history:
                self.history_text.insert(END, f"{item[1]}\n")
            self.history_text.config(state=DISABLED)  # Set state back to
disabled to make it read-only
        else:
            self.history_text.config(state=NORMAL)  # Set state to normal
to allow modifications
            self.history_text.delete(1.0, END)
            self.history_text.insert(END, "No history available.")
            self.history_text.config(state=DISABLED)  # Set state back to
disabled to make it read-only

# Execution
if __name__== '_main_':
    # Create the main window of an application
    root = Tk()
    # Tell our calculator class to use this window
    my_gui = Calculator(root)
    # Executable loop on the application, waits for user input
    root.mainloop()
```

RESULTS AND CONCLUSION

Conclusion:

In conclusion, the scientific calculatory offers a robust toolset for performing complex calculations across various scientific disciplines. Its functionality, accuracy, and versatility make it an indispensable asset for researchers, engineers, educators, and students alike. By harnessing advanced mathematical algorithms and intuitive user interfaces, the scientific calculatory streamlines workflows and enhances problem-solving capabilities. As technology continues to evolve, the scientific calculatory remains a cornerstone in advancing scientific inquiry, innovation, and discovery.

Results:



REFERENCES

Tkinter Documentation: https://docs.python.org/3/library/tkinter.html

W3Schools Tkinter Tutorial: https://www.w3schools.com/python/python_tkinter.asp

JetBrains PyCharm: https://www.jetbrains.com/pycharm/