Project
FCCS Department
December 2021

GUI Simple Calculator

CS111 Final Project

Haya Baqais 3910002

Dept. of Forensic Computing and Cyber Security
Faculty of Computer Science and Information Technology
University of Prince Mugrin, Madinah KSA

ABSTRACT

This report will elaborate the details of the implementation and the design of a user-friendly Graphical User Interface (GUI) calculator to satisfy CS111 project requirement. The project focused on creating a simple calculator using Tkinter library which expand the acceptable end-users' characteristics. The main goal of the project was to build a system that can process several mathematical operations and return accurate results. Due to the project goal, it will ease the mathematical calculations for large number of users. Besides, it will reduce the calculation errors and the time required for solving mathematical problems.

Table of Contents

INTRODUCTION

On recent days, computers are considered an important part from any office whether it belongs to a student or an employee. Therefore, almost everyday new tools are showing to automate office work. One of the most well-known tools are calculators. Calculators are software which are designed to do the mathematical processing where the most human mistakes are shown. The aim of this project is to automate this part of office work to make it easier to get an accurate result of mathematical operations in the least amount of time.

SOFTWARE REQUIREMENT SPECIFICATION

Business Value

Conservative estimates of tangible value to the users includes:

1. Ease mathematical calculations
2. Ensure the accuracy of the results
3. Minimize the time of calculation

Special Issues or Constraints

Conservative estimates of special issues this software solves includes:

1. Lack of knowledge about mathematical calculations
2. Human mistakes during calculating
3. Times needed to operate large number calculations

Requirement Analysis

Functional Requirements

- User can enter numbers (Integers and floats)
- User can change the number sign
- User can choose mathematical operation
- User can compute the equation result.
- User can clear the equation

Non-Functional Requirements

- Proper exception handling
- User-friendly Graphical User Interface
- Limit the input of the user to the accepted values

Features: work - not work why? - future ideas

This section elaborates the features, status, and their implementation on the project. Besides, it adds some possible future features that can enhance the functionalities of the project.

Feature 1: Graphical User Interface (GUI)

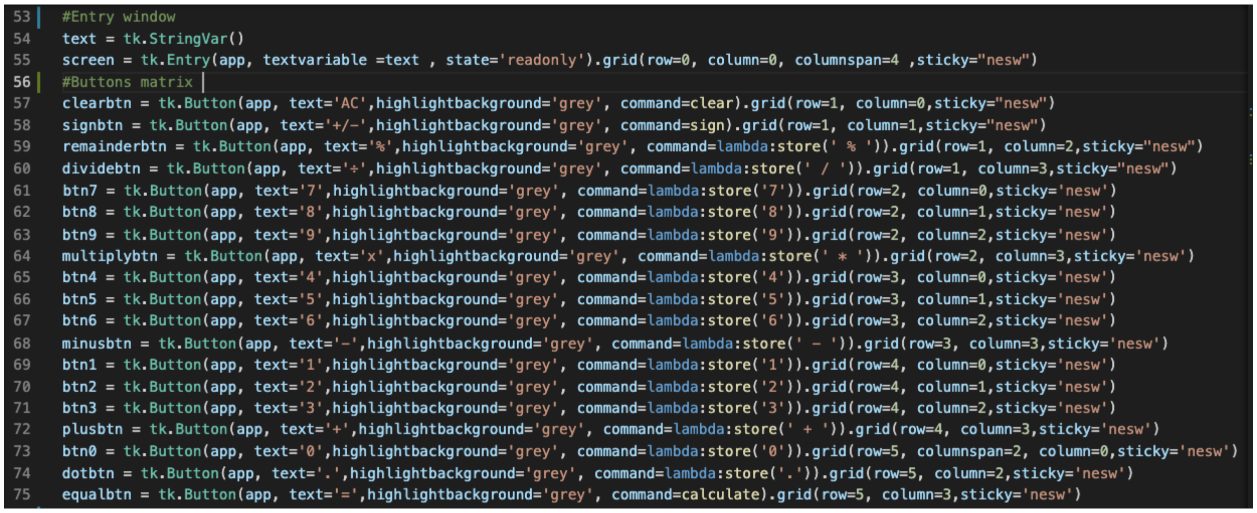
Description:

This feature is providing Graphical User Interface which is a simple, pretty, and user-friendly interface that interacts with the user as a replacement for the console. The feature is expanded to make sure that the interface will keeps its structure even if the geometry got changed.

Status: The feature was implemented, and it is working.

Code Snipped:

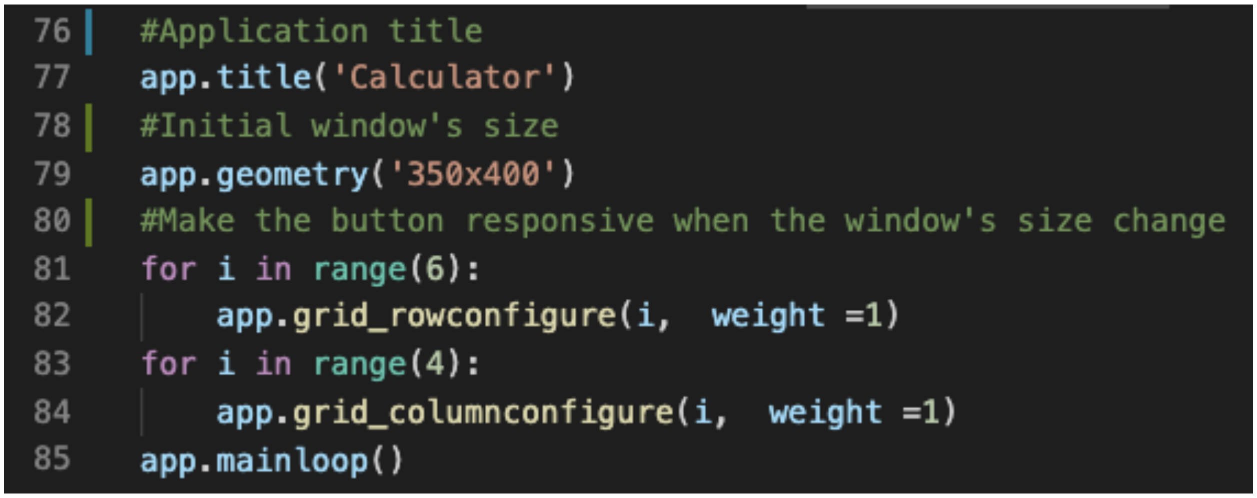
- Basic window’s layout code



```python
#Entry window
text = tk.StringVar()
screen = tk.Entry(app, textvariable =text , state='readonly').grid(row=0, column=0, columnspan=4 ,sticky="nesw")
#Buttons matrix
clearbtn = tk.Button(app, text='AC',highlightbackground='grey', command=clear).grid(row=1, column=0,sticky="nesw")
signbtn = tk.Button(app, text='+/-',highlightbackground='grey', command=sign).grid(row=1, column=1,sticky="nesw")
remainderbtn = tk.Button(app, text='%',highlightbackground='grey', command=lambda:store(' % ')).grid(row=1, column=2,sticky="nesw")
dividebtn = tk.Button(app, text='÷',highlightbackground='grey', command=lambda:store(' / ')).grid(row=1, column=3,sticky="nesw")
btn7 = tk.Button(app, text='7',highlightbackground='grey', command=lambda:store('7')).grid(row=2, column=0,sticky='nesw')
btn8 = tk.Button(app, text='8',highlightbackground='grey', command=lambda:store('8')).grid(row=2, column=1,sticky='nesw')
btn9 = tk.Button(app, text='9',highlightbackground='grey', command=lambda:store('9')).grid(row=2, column=2,sticky='nesw')
multiplybtn = tk.Button(app, text='x',highlightbackground='grey', command=lambda:store(' * ')).grid(row=2, column=3,sticky='nesw')
btn4 = tk.Button(app, text='4',highlightbackground='grey', command=lambda:store('4')).grid(row=3, column=0,sticky='nesw')
btn5 = tk.Button(app, text='5',highlightbackground='grey', command=lambda:store('5')).grid(row=3, column=1,sticky='nesw')
btn6 = tk.Button(app, text='6',highlightbackground='grey', command=lambda:store('6')).grid(row=3, column=2,sticky='nesw')
minusbtn = tk.Button(app, text='-',highlightbackground='grey', command=lambda:store(' - ')).grid(row=3, column=3,sticky='nesw')
btn1 = tk.Button(app, text='1',highlightbackground='grey', command=lambda:store('1')).grid(row=4, column=0,sticky='nesw')
btn2 = tk.Button(app, text='2',highlightbackground='grey', command=lambda:store('2')).grid(row=4, column=1,sticky='nesw')
btn3 = tk.Button(app, text='3',highlightbackground='grey', command=lambda:store('3')).grid(row=4, column=2,sticky='nesw')
plusbtn = tk.Button(app, text='+',highlightbackground='grey', command=lambda:store(' + ')).grid(row=4, column=3,sticky='nesw')
btn0 = tk.Button(app, text='0',highlightbackground='grey', command=lambda:store('0')).grid(row=5, columnspan=2, column=0,sticky='nesw')
dotbtn = tk.Button(app, text='.',highlightbackground='grey', command=lambda:store('.')).grid(row=5, column=2,sticky='nesw')
equalbtn = tk.Button(app, text='=',highlightbackground='grey', command=calculate).grid(row=5, column=3,sticky='nesw')
```

Figure 1: Screen’s layout code

- Title, geometry, and responsiveness code



```python
#Application title
app.title('Calculator')
#Initial window's size
app.geometry('350x400')
#Make the button responsive when the window's size change
for i in range(6):
    app.grid_rowconfigure(i,  weight =1)
for i in range(4):
    app.grid_columnconfigure(i,  weight =1)
app.mainloop()
```

Figure 2: Title, geometry, and responsiveness code

Feature 2: Clear equation

Description:

This function is providing clear feature. Clearing the entry and the equation on the cache makes the user able to start new calculations and avoid possible confusion from processing sequence of separate equations. The application is relying on two main variables which are *equation* and *result*. *Equation* variable contains the full equation to be computed on further steps. *Result* variable stores the result after certain computing process. To achieve clearing functionality, both variables are assigned to empty values. Besides, the entry screen is being set to 0. This function is being triggered when the "AC" button is being clicked.

Status: The feature was implemented, and it is working.

Pseudocode:

```
START
SET equation = “”
SET result=””
SET text=0
END
```

Flow Chart:

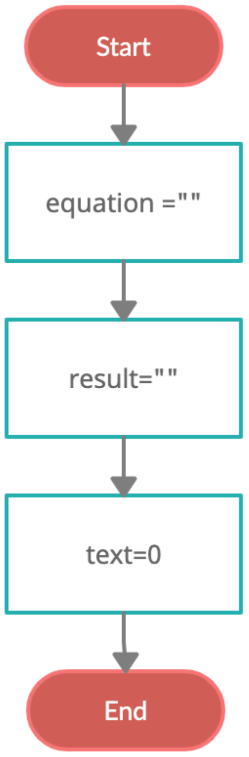
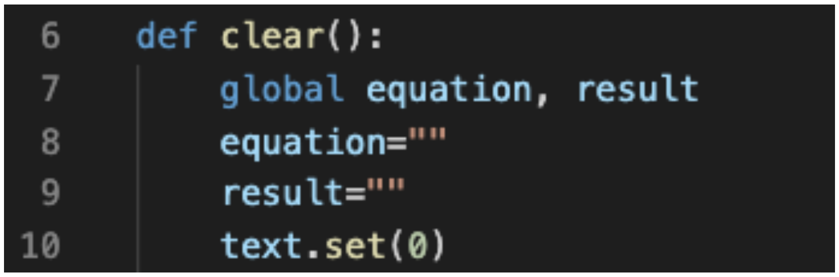


Figure 3: Clear equation flowchart

Code Snipped:



```python
def clear():
    global equation, result
    equation=""
    result=""
    text.set(0)
```

Figure 4: Clear function

Feature 3: Change number sign

Description:

This function is providing change a number sign feature. Changing number sign expands the range of the number that can be used for further calculations. This can be implemented by taking the required number from the equation, multiply it with -1, and replace the original value with the result. This function can be triggered by clicking "+/-" button.

Status: The feature was implemented, and it is working.

Pseudocode:

```
START
TRY:
        SET tmp = text.get()
        IF tmp !=0 THEN
                Result = int(tmp) * -1
                Equation = equation.replace(tmp,str(result))
                SET text=result
        ENDIF
EXCEPT:
        SET text=”Error”
        Result=””
        Equation=””
END
```

Flow Chart:

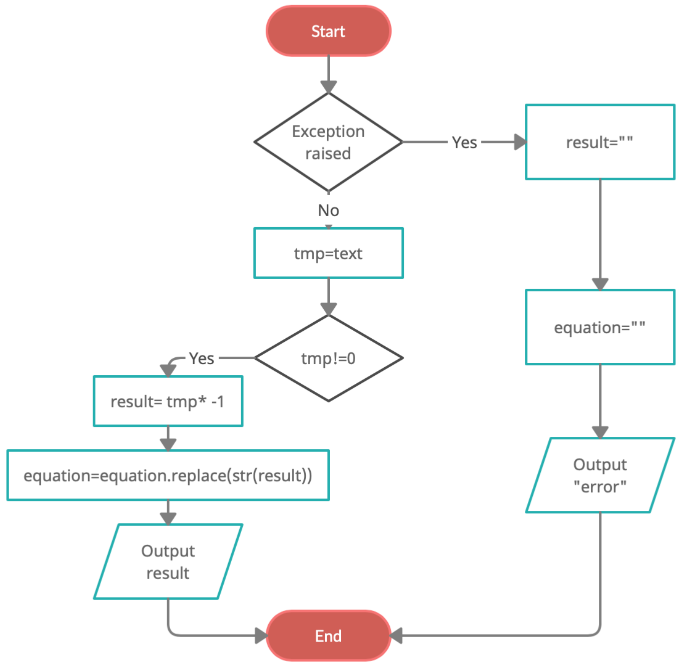


Figure 5: Change sign function flow chart 1

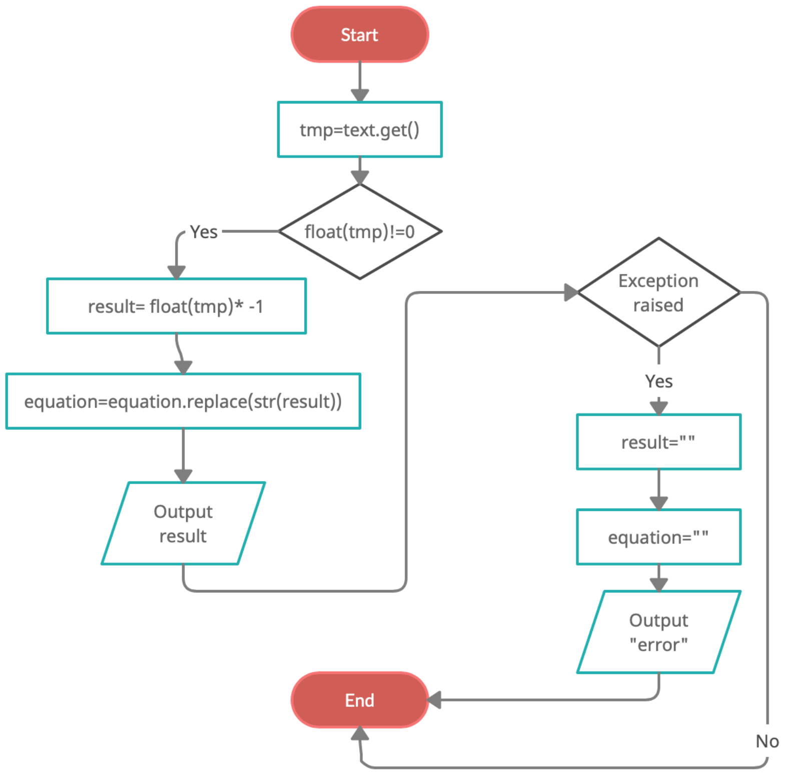
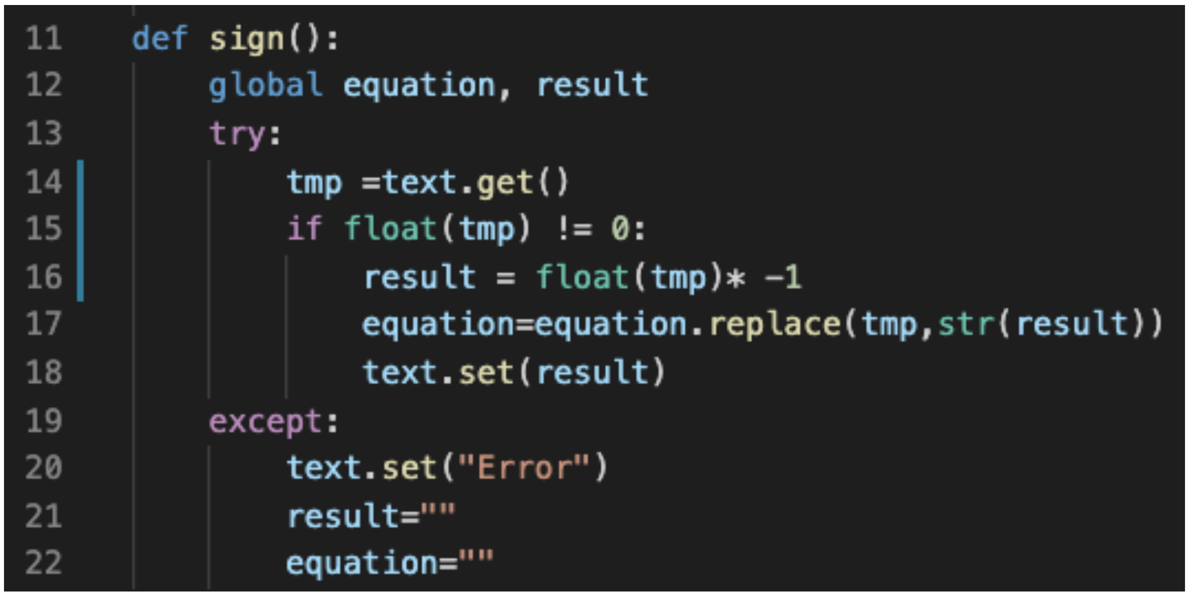


Figure 6: Change sign function flow chart 2

Code Snipped:



```python
def sign():
    global equation, result
    try:
        tmp =text.get()
        if float(tmp) != 0:
            result = float(tmp)* -1
            equation=equation.replace(tmp,str(result))
            text.set(result)
    except:
        text.set("Error")
        result=""
        equation=""
```

Figure 7: Change sign function

Feature 4: Add values

Description:

This function is providing store new values to the equation for further computation. Storing new values properly to the equation will ensure the accuracy of the calculations. This can be implemented by passing the value of the clicked button to the storing function which will handle adding and reflecting the required result on the entry screen. Using dedicated button’s value limits the error that can be occurred from invalid user input.

Status: The feature was implemented, and it is working.

Pseudocode:

```
START
TRY:
    INPUT num
    IF not num.isnumeric() and equation==”” and result==””:
        Equation=”0”+num
    Elif not num.isnumeric() and equation==”” and result!=””:
        Equation = result+num
    Elif len(equation.split())==3 and not num.isnumeric():
        Jump Calculate()
        Equation =result+num
    Elif num.isnumeric():
        Equation+=num
        If len(equation.split())==1:
            Text.set(equation.split()[0])
        Elif len(equation.split())==3:
            Text.set(equation.split()[2])
    Else:
        Equation+=num
EXCEPT:
    SET text=”Error”
    Result=””
    Equation=””
END
```

Flow Chart:

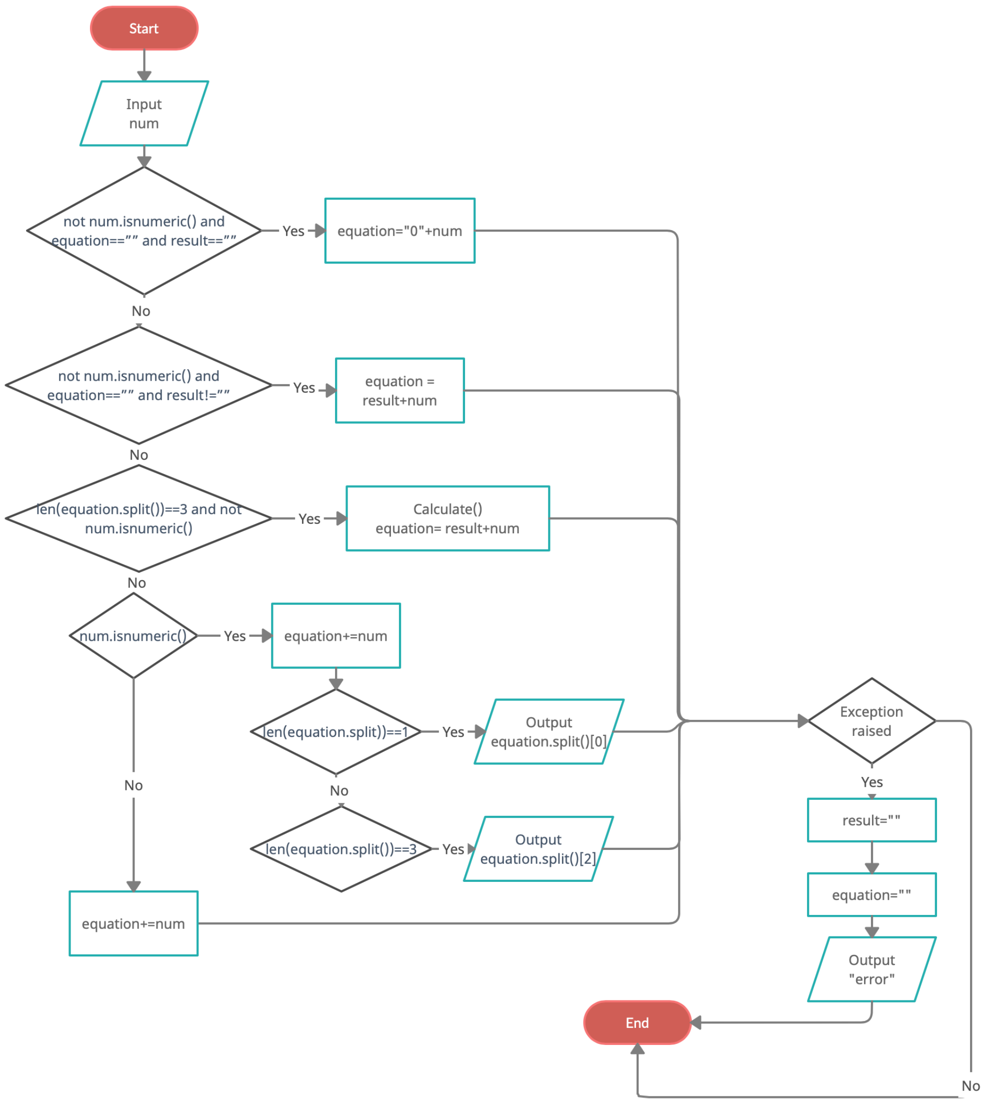
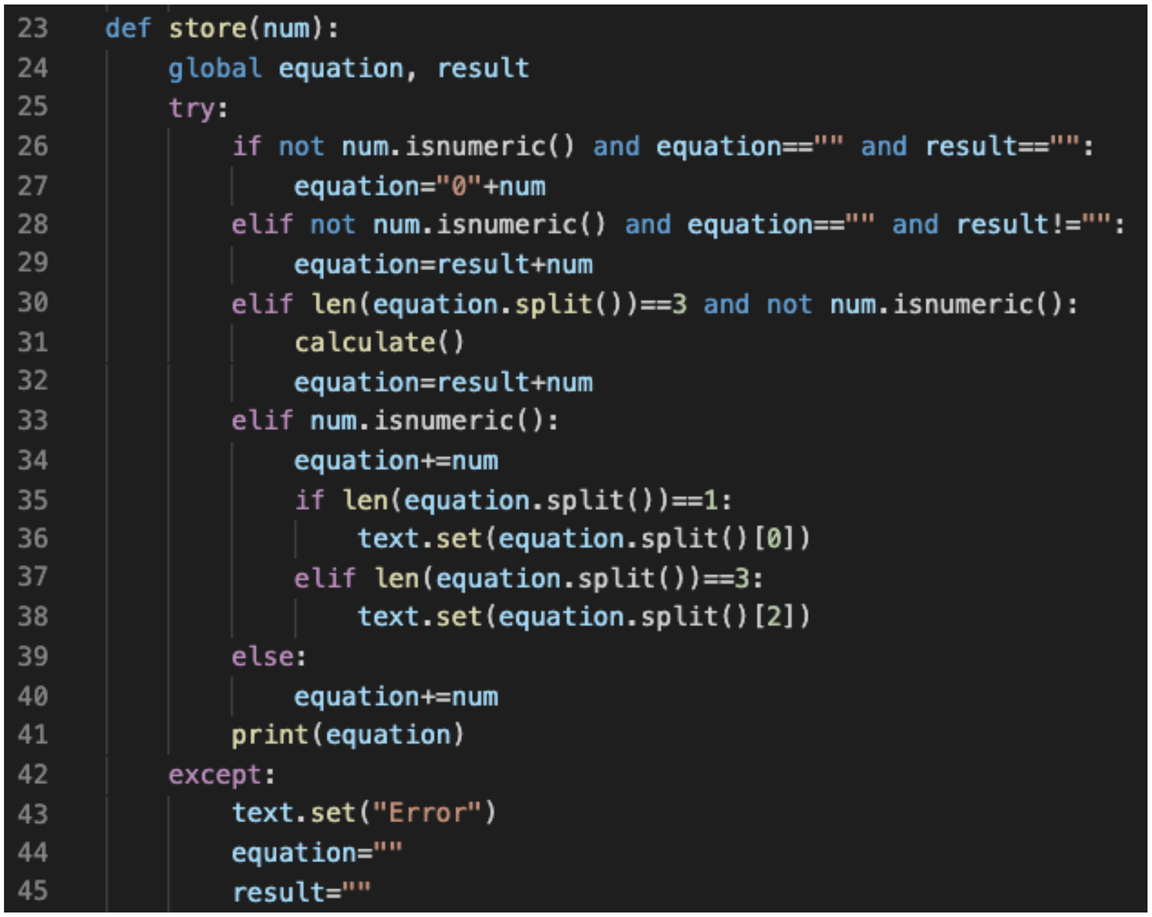

Start
Input num
not num.isnumeric() and equation=="" and result==""
Yes
equation="0"+num
No
not num.isnumeric() and equation=="" and result!=""
Yes
equation = result+num
No
len(equation.split())==3 and not num.isnumeric()
Yes
Calculate() equation= result+num
No
num.isnumeric()
Yes
equation+=num
len(equation.split())==1
Yes
Output equation.split()[0]
No
len(equation.split())==3
Yes
Output equation.split()[2]
No
equation+=num
Exception raised
Yes
result=""
equation=""
Output "error"
End
No

Figure 8: Add value function flow chart

Code Snipped:



```python
def store(num):
    global equation, result
    try:
        if not num.isnumeric() and equation=="" and result=="":
            equation="0"+num
        elif not num.isnumeric() and equation=="" and result!="":
            equation=result+num
        elif len(equation.split())==3 and not num.isnumeric():
            calculate()
            equation=result+num
        elif num.isnumeric():
            equation+=num
            if len(equation.split())==1:
                text.set(equation.split()[0])
            elif len(equation.split())==3:
                text.set(equation.split()[2])
        else:
            equation+=num
        print(equation)
    except:
        text.set("Error")
        equation=""
        result=""
```

Figure 9: Add value function

Feature 5: Compute result

Description:

This function is providing evaluate the result of the equation. This feature considered the backbone of any calculator as it is supposed to give the result of mathematical operations. This can be implemented by evaluation the mathematical operation that have been stored into *equation* variable.

Status: The feature was implemented, and it is working.

Pseudocode:

```
START
TRY:
        IF equation =”” and result ==””:
                SET text=”0”
        ELSE:
                Result=str(eval(equation))
                SET text=result
                Equation =””
EXCEPT:
        SET text=”Error”
        Result=””
        Equation=””
END
```

Flow Chart:

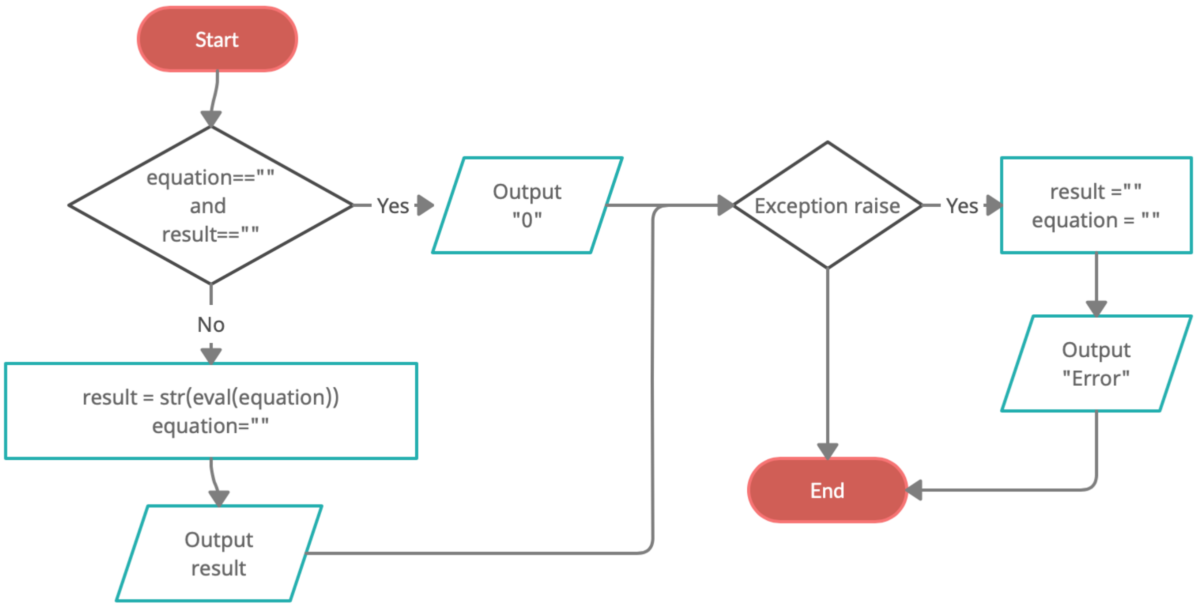
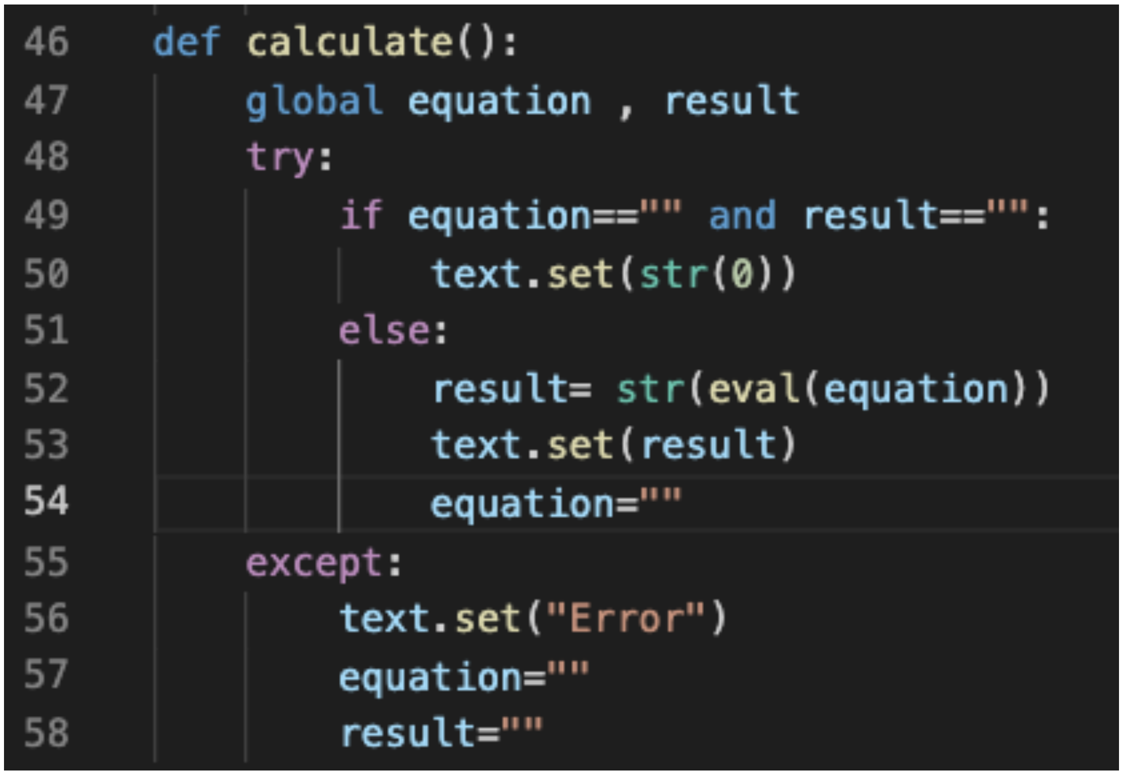


Figure 10: Calculate function flow diagram

Code Snipped:



```
def calculate():
    global equation , result
    try:
        if equation=="" and result=="":
            text.set(str(0))
        else:
            result= str(eval(equation))
            text.set(result)
            equation=""
    except:
        text.set("Error")
        equation=""
        result=""
```

Figure 11: Calculate function

Future Features

This project can be expanded to cover more operations categorized into different mode. First mode can be an expansion mathematical calculation where it covers basics, trigonometry, exponent, log, and root. Second mode can be programmer mode where it covers the conversion between known bases. Thirds mode can be graphs mode where it calculates the points of an equation.

DESIGN CONSIDERATIONS

Design Constraints

Hardware

- PC or Laptop

Software

- Operating System
- Python 3.10
- Tkinter package

User Interface

The application consists of a single interface that interact with the end user. As shown in figure below, the entry screen will show the current user input and the computed result followed by a matrix of button where each one will trigger certain functionality. As an exception handling, all failures on the function will print "Error" on the entry screen and empty bot the equation and the result variables.

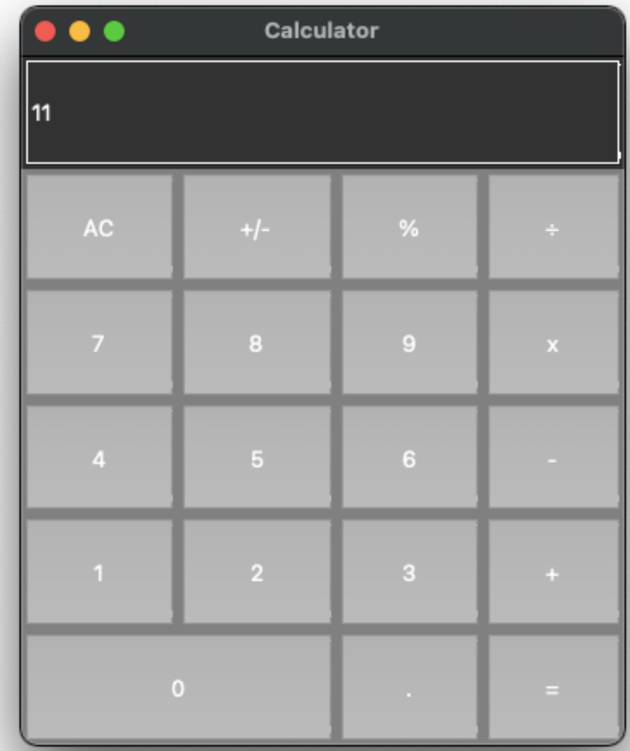


Figure 12: Valid User Input

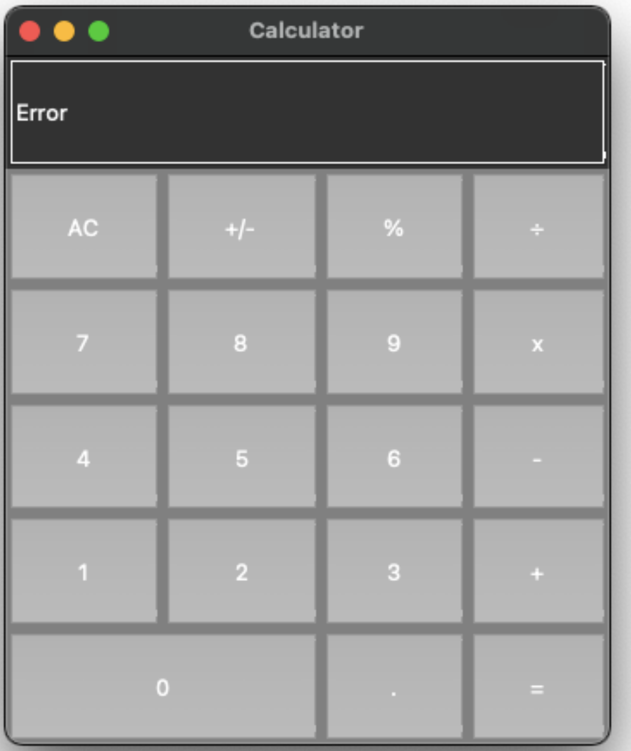


Figure 13: Error message for invalid input

REFERENCE

GeeksforGeeks. (2021, March 28). *Python Tkinter Tutorial*. GeeksforGeeks. Retrieved December 2021, from https://www.geeksforgeeks.org/python-tkinter-tutorial/.

GeeksforGeeks. (2021, June 30). *Python | Simple GUI calculator using Tkinter*. GeeksforGeeks. Retrieved December 2021, from https://www.geeksforgeeks.org/python-simple-gui-calculator-using-tkinter/.

APPENDIX A: SOURCE CODE

Code was uploaded on github for efficiency and readability on the following link:
https://github.com/hayabaq/cs111-project