Middlesex University Mauritius

3/28/2021

CST1500 Computer Systems Architecture and Operating Systems

Python Coursework Calculator

Avinesh Culloo (M00776456)
Muhammad Hassan Maudarbocus (M00775610)

Contents

Introduction

This project is based on creating a fully functional GUI calculator using Python libraries. The calculator allows a user to carry out both basic and scientific calculations.

Basic calculations include the following mathematic expressions:

1) Addition and Subtraction
2) Multiplication and Division
3) Equal and Clear

Besides, the scientific calculations that can be carried out are:

1) Square Root
2) Natural logarithms
3) Make use of constants (e.g π, e)
4) Exponentiation
5) Modulus (abs)

Python

Python is a high-level programming language that can be used for the following applications:

1) Web Development: Django, Pyramid, Bottle, Tornado, Flask, web2py
2) GUI Development: tkinter, PyGObject, PyQt, PySide, Kivy, wxPython
3) Scientific and Numeric: SciPy, Pandas, IPython
4) Software Development: Buildbot, Trac, Roundup

For this coursework, tkinter was used for the GUI development.

Tkinter

Tkinter is not the only GUI Programming toolkit for python as mentioned above, but it is the most commonly used one. It provides a powerful object-oriented interface to the Tk GUI toolkit.

Tkinter was imported by running the following code as shown in fig1:

fig 1

```
from tkinter import *
```

Math Library

Math library in python allows us to perform a more complex calculation in python which can be imported using the code as shown in fig 2 below.

fig 2

```
import math
```

Implementation

Object-Oriented Programming (OOP)

OOP is a computer programming model that organizes software design around data, or objects rather than functions or logic.
Since students had to work in pairs (collaborative programming), OOP allows each student to implement a part of the application to achieve the desired result.

OOP was implemented as shown in fig 3:

```
# class to start the application
class App(Tk):
    # initialise constructor
    def __init__(self):
        super(App, self).__init__()
        self._frame = None
        self.title("Calculator")
        self.switchFrame(Standard)

    # function to switch Frame
    def switchFrame(self, class_Frame):
        newFrame = class_Frame(self)
        if self._frame is not None:
            self._frame.destroy()

        self._frame = newFrame
        self._frame.pack()


class Standard(Frame):
    def __init__(self, master):
        Frame.__init__(self, master)
```

fig 3

OOP allows:

1. Switch frames by destroying the old one
2. Set the window title
3. Switch between the standard and scientific calculator.

Tkinter

1) Basic Widgets

- Frame

It is a rectangular region used to group related widgets or provide padding between widgets. A snippet is shown in fig 4.

Fig 4

```
# Frame for display
display_frame = Frame(self, width=370, height=70, bd=0, highlightcolor="LightSkyBlue3",
                      highlightbackground="gray", highlightthickness=2)
```

- Button
  It contains text and calls a function when clicked. A snippet is shown in fig 5.

Fig 5

```
Button(button_Frame, text="SCI", width=10, height=3, bg="DarkOrange4",
       font=('Serif', 10), command=lambda: master.switchFrame(Scientific)).grid(
    row=4, column=0, padx=1, pady=1)
```

- Entry
  It allows a single line of text. A snippet is shown in fig 6.

Fig 6

```
input_Field = Entry(display_frame, font=('Serif', 18, 'bold'), textvariable=inputOutput_Text, width=50,
                    bg="light grey",
                    bd=15,
                    justify=RIGHT)
```

- Label
  It is used to display text on the screen. A snippet is shown in fig 7.

Fig 7

```
Label(History_Frame, text=row, font=('Serif', 12), bg="azure2").place(x=25, y=yAxis)
```

2) Geometry Manager
   - Pack: It packs the widget on top of the other. Implementation is shown in fig 8.

     Fig 8

     ```
     display_frame.pack(side=TOP)
     ```

   - Grid: It splits either a window or frame into rows and columns. Implementation is shown in fig 9.

     Fig 9

     ```
     scientific.grid(row=4, column=0, padx=1, pady=1)
     ```

   - Place: It places the widget at specific positions designed by the programmer. Implementation is shown in fig 10.

fig 10

```
Button(History_Frame, text="BACK", font=('Serif', 10), command=lambda: master.switchFrame(Calculator),
       bg="wheat3").place(x=100, y=210)
```

Functions

Functions were used to carry out specific actions when the buttons are clicked. For example, in fig 11 below, when equal (=) in the calculation is clicked, the code in the function button_equal() will be executed.

Fig 11

```
def button_equal():
    global calculation
    result = str(eval(calculation))
    inputOutput_Text.set(result)
    calculation = ""
```

Lists

List was used to storing several items in a single variable. It helps to reduce the use of several variables. The code is shown in fig 9 below.

```
# list to store button names
button_names = ["π", "e^", "CE", "Abs(",
                "^2", "(", ")", "x10^(",
                "√(", "^", "ln(", "/",
                "7", "8", "9", "*",
                "4", "5", "6", "-",
                "1", "2", "3", "+",
                "⌫", "0", "•", "="]

# list to store button syntax
button_syntax = ["math.pi", "(math.e)**", "", "abs(",
                 "**2", "(", ")", "*10**(",
                 "math.sqrt(", "**", "math.log(", "/",
                 "7", "8", "9", "*",
                 "4", "5", "6", "-",
                 "1", "2", "3", "+",
                 "", "0", ".", ""]
```

Fig 9

For Loop

For loop was used to loop through the lists to extract its contents. The code is shown in fig 10 below.

Fig 10

```
for row in range(0, 3):
    for column in range(0, 3):
```

Error Handling

This will prevent the program from crashing unexpectedly. The code is shown in fig 11 below.

Fig 11

```
try:
    # calculate the result by using eval
    result = str(eval(calculation))
    output_text.set(result)
    history.append(f"{expression} = {result}")
    calculation = ""  # set calculation to default
    expression = "0"
except (ZeroDivisionError, SyntaxError):
    output_text.set("ERROR")  # display errors if divide by zero or syntax error
```

Some screenshots of the calculator.

1.Fig 12 is showing the calculator when launched. 2. Fig 13 is showing the basic calculation.

Fig 12

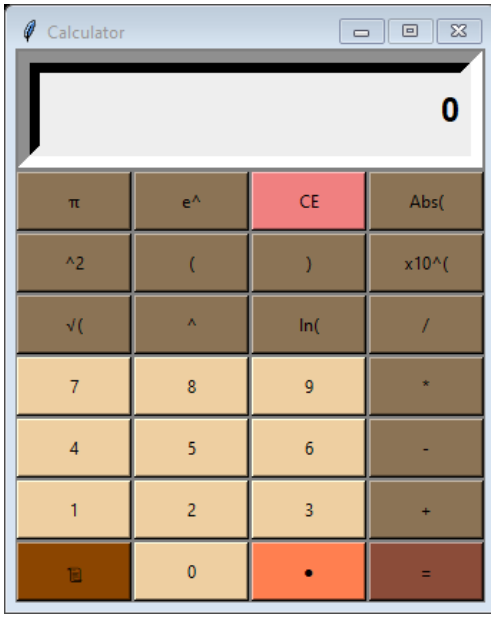
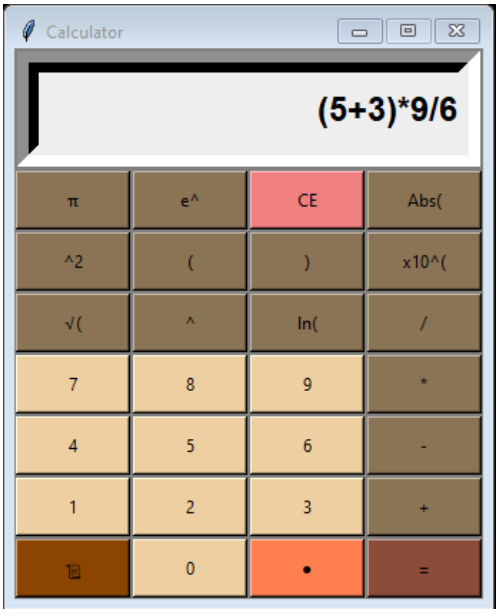
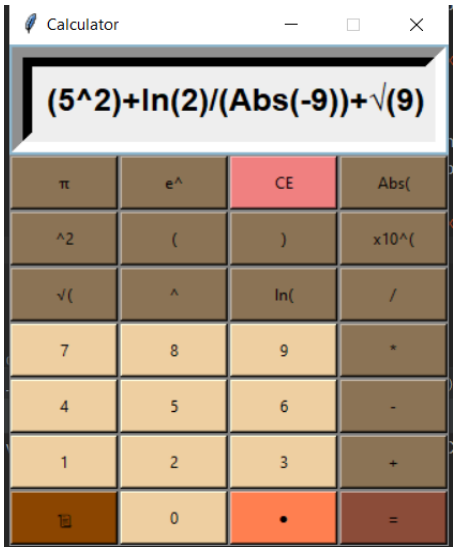


fig 13



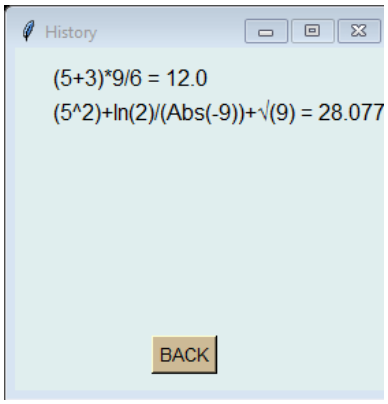
3. Fig 14 is showing some advanced calculations.

Fig 14



4. There is a history button to allow the user to view past calculation as shown in fig 15.

Fig 15



How can it be improved?

1) Allow moving through the expression to make any changes.
2) Enable the user to clear the last item appended to the expression.

Special thanks to:

Mr. Divesh Roopowa, ***Senior GTA in IT/CSSE*** who was always there to help us with our queries and guide us on the right path.

References:

- https://www.python.org/about/apps/ [Accessed 16 March 2021].