Calculator Using PyQt5 in Python

Darshil Mavadiya(22BEC508)
Dept. of Electronics and Communication Engineering
Institute of Technology,Nirma University
Ahmedabd,India
22bec508@nirmauni.ac.in

Anant Kothivar(22BEC506)
Dept. of Electronics and Communication Engineering
Institute of Technology,Nirma University
Ahmedabad, India
22bec506@nirmauni.ac.in

***Abstract*—One Application that we all use on a daily basis is the calculator. As we already knew how to write a calculator program in linux command line, We constructed and presented the calculator using Python Graphical User Interface in this report. Python includes a number of tools for creating a graphical user interface, including Tkinter, wxPython, PyQt, and PySide2. For the implementation in this report, we used the PyQt library.**

***Index Terms*—Calculator Project, Graphical User Interface(GUI), Python, PyQt.**

I. INTRODUCTION

Python provides a number of alternatives for building a GUI (Graphical User Interface). PyQt is the most often used GUI technique out of all the options. As a result, PyQt5 is utilised in this work to create a simple calculator application with a graphical user interface. The following are the main outcomes by this program:

- An screen to display the click actions and the result.
- Buttons with numeric values and be clickable.
- Perform mathematical operations (addition, subtraction, multiplication, division).
- Perform equation calculation (=) and clearing the screen (c).

II. ABOUT PYQT

PyQt5 is a cross-platform GUI toolkit that includes Python bindings for the Qt v5 framework. Because of the capabilities and simplicity given by this toolkit, developing an interactive desktop application is a simple. PyQt is licensed under the GNU GPL v3 and the Riverbank Commercial License on all supported platforms.

PyQt is a Python binding for Qt, a set of C++ libraries and development tools that includes platform-independent abstractions for Graphical User Interfaces (GUIs), threads, networking, regular expressions, XML, SQL databases, and many other sophisticated features.

PyQt is available in two editions:PyQt4 and PyQt5. we have used PyQt5 in our program.

III. IMPLEMENTATION

In this section, logic and implementation of our program is presented.

A. Importing required libraries

```
#!/usr/bin/env python3

import sys
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QGridLayout
from PyQt5.QtWidgets import QLineEdit
from PyQt5.QtWidgets import QPushButton
from PyQt5.QtWidgets import QVBoxLayout
from functools import partial
from PyQt5.QtWidgets import QApplication
from PyQt5.QtWidgets import QMainWindow
from PyQt5.QtWidgets import QWidget

```

Fig. 1.

B. Logic of the program

The Model-View-Controller design pattern was used to develop our calculator program. There are three levels of code in this pattern, each having its own function:

- The Model - Our calculator's computations will be handled by the model.
- The View - The view is responsible for the app's graphical user interface. The view for our calculator will be the window on the screen.
- The Controller - The controller connects the model with the view to make the application work. The controller will receive user events from the calculator's GUI, instruct the model to do calculations, and display the results on the GUI.

Here is the logic for our GUI desktop application's Steps-by-Step.

1) On the view, the user makes a request (event).
2) The controller is notified of the user's action by the view.
3) The controller receives the user's request and requests a response from the model.
4) The model interprets the controller query, executes the necessary procedures, and provides a response.
5) When the controller receives the model's response, it updates the view.
6) On the display, the user finally sees the expected result.

C. Creating the basic structure

The code shown below is what any basic graphical user interface program needs to execute.

```
17
18 class PyCalcUi(QMainWindow):
19     def __init__(self):
20         super().__init__()
21         self.setWindowTitle('My Calculator')
22         self.setFixedSize(470, 470)
23         self.generalLayout = QVBoxLayout()
24         self._centralWidget = QWidget(self)
25         self.setCentralWidget(self._centralWidget)
26         self._centralWidget.setLayout(self.generalLayout)
27         self._createDisplay()
28         self._createButtons()
```

Fig. 2.

```
78 def main():
79     pycalc = QApplication(sys.argv)
80     view = PyCalcUi()
81     view.show()
82     model = evaluateExpression
83     PyCalcCtrl(model=model, view=view)
84     sys.exit(pycalc.exec_())
85
```

Fig. 3.

PyCalcUi is the class that creates the GUI. QMainWindow is a subclass of this class. On line 21, the window's title is set to PyCalc. The setFixedSize() function is used to restrict a window's size and prevent it from being resized. A QWidget object is created on line 24 to operate as the central widget. This object will be parent for the rest of the GUI Component.

Figure 3 illustrates the basic method of our program. QApplication() generates the object pycalc in the main method. The GUI is then displayed using view.show(). pycalc.exec_() is used to run the application's event loop at the end of the main method .

D. Creating the view

The display and controls for the numbers, basic mathematical operators, and clearing the display were added here.

```
29
30     def _createDisplay(self):
31         self.display = QLineEdit()
32         self.display.setFixedHeight(50)
33         self.display.setAlignment(Qt.AlignRight)
34         self.display.setReadOnly(True)
35         self.generalLayout.addWidget(self.display)
36
```

Fig. 4.

For the calculator's general layout, we used a QVBoxLayout. To layout the buttons, we used a QGridLayout object. QLineEdit is used for the display, while QPushButton is used for the buttons. The display is at the top, and the buttons are in a grid layout at the bottom, due to a QVBoxLayout. We used a QLineEdit object to create the display widget.

```
37     def _createButtons(self):
38         self.buttons = {}
39         buttonsLayout = QGridLayout()
40         buttons = {'7': (0, 0),
41                    '8': (0, 1),
42                    '9': (0, 2),
43                    '/': (0, 3),
44                    'C': (0, 4),
45                    '4': (1, 0),
46                    '5': (1, 1),
47                    '6': (1, 2),
48                    '*': (1, 3),
49                    '(': (1, 4),
50                    '1': (2, 0),
51                    '2': (2, 1),
52                    '3': (2, 2),
53                    '-': (2, 3),
54                    ')': (2, 4),
55                    '0': (3, 0),
56                    '00': (3, 1),
57                    '.': (3, 2),
58                    '+': (3, 3),
59                    '=': (3, 4),
60                   }
61         for btnText, pos in buttons.items():
62             self.buttons[btnText] = QPushButton(btnText)
63             self.buttons[btnText].setFixedSize(60, 60)
64             buttonsLayout.addWidget(self.buttons[btnText], pos[0], pos[1])
65         self.generalLayout.addLayout(buttonsLayout)
66
```

Fig. 5.

```
67     def setDisplayText(self, text):
68         self.display.setText(text)
69         self.display.setFocus()
70
71     def displayText(self):
72         return self.display.text()
73
74     def clearDisplay(self):
75         self.setDisplayText('')
76
```

Fig. 6.

E. Creating the Controller

here calculator's controller class is shown in below code:

```
86
87 class PyCalcCtrl:
88     def __init__(self, model, view):
89         self._evaluate = model
90         self._view = view
91         self._connectSignals()
92
93     def _calculateResult(self):
94         result = self._evaluate(expression=self._view.displayText())
95         self._view.setDisplayText(result)
96
97     def _buildExpression(self, sub_exp):
98         if self._view.displayText() == ERROR_MSG:
99             self._view.clearDisplay()
100        expression = self._view.displayText() + sub_exp
101        self._view.setDisplayText(expression)
102
103    def _connectSignals(self):
104        for btnText, btn in self._view.buttons.items():
105            if btnText not in {'=', 'C'}:
106                btn.clicked.connect(partial(self._buildExpression, btnText))
107        self._view.buttons['='].clicked.connect(self._calculateResult)
108        self._view.display.returnPressed.connect(self._calculateResult)
109        self._view.buttons['C'].clicked.connect(self._view.clearDisplay)
110
```

Fig. 7.

```
112 def evaluateExpression(expression):
113     try:
114         result = str(eval(expression, {}, {}))
115     except Exception:
116         result = ERROR_MSG
117
118     return result
119
```

Fig. 8.

The view as well as the model will be connected by this class.

The controller class is used to have the calculator react to user events by performing actions. This will ensure that your calculator function correctly.

this controller class performs three main tasks:

1) Access the GUI's public interface
2) Handle the creation of math expressions
3) Connect button clicked signals with the appropriate slots

F. *Implementing the Model*

The logic is handled by the model, which is a layer of code. The logic in this case is based on basic math calculations. The math expressions entered by user will be evaluated using this model.

```
92
93      def _calculateResult(self):
94          result = self._evaluate(expression=self._view.displayText())
95          self._view.setDisplayText(result)
96
97      def _buildExpression(self, sub_exp):
98          if self._view.displayText() == ERROR_MSG:
99              self._view.clearDisplay()
100         expression = self._view.displayText() + sub_exp
101         self._view.setDisplayText(expression)
102
```

Fig. 9.

We've used eval() to evaluate a string as an expression in this case. If the string is an expression, the result will be returned. Otherwise, an error message will be displayed.

IV. Result

here in this section, output view of our calculator is shown. also one test input with its output is shown.

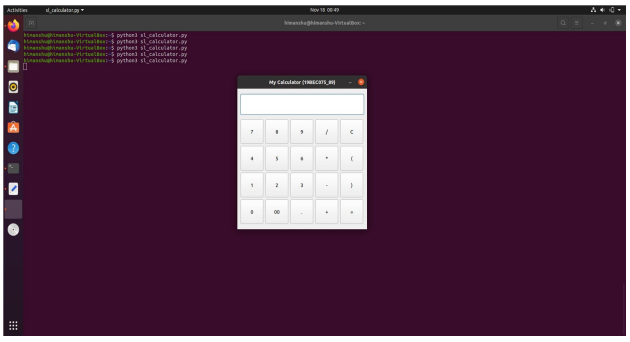

Fig. 10.

A. *Test Input*

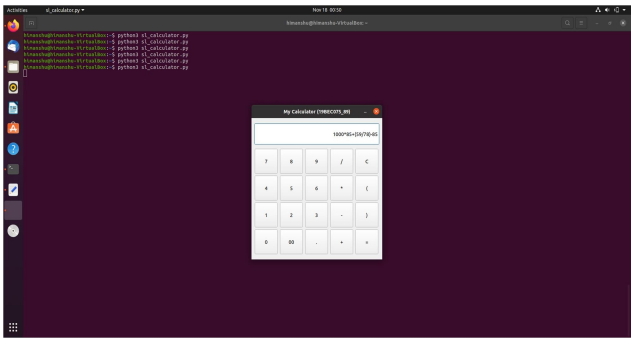

Fig. 11.

B. *Test Output*

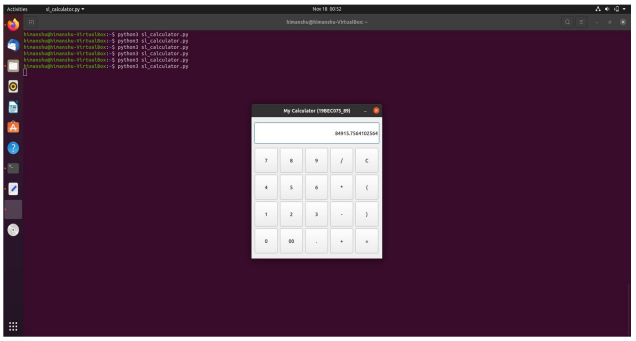

Fig. 12.

V. Conclusion

Desktop Application with a Graphical user interface (GUI) still account for a significant portion of the software development market. Python includes a number of frameworks and packages that can be used creating Modern and reliable GUI applications. We constructed a basic calculator in this paper using PyQt, which is undoubtedly one of the most popular and reliable frameworks for developing GUI desktop applications in Python. we implemented logic and GUI of the calculator.

Acknowledgment

We would like to express our gratitude to Dr.Vaishali Dhare who provided us the opportunity to make research on the topic of our interest and present it in form of paper.

References

[1] "PyQt5 Tutorial." PyQt5 Tutorial. www.tutorialspoint.com. Accessed November 18, 2021. https://www.tutorialspoint.com/pyqt5/index.htm.

[2] "PyQt/Tutorials - Python Wiki." PyQt/Tutorials - Python Wiki. wiki.python.org. Accessed November 18, 2021.https://wiki.python.org/moin/PyQt/Tutorials.

[3] Parwiz. "PyQt5 How To Make GUI Calculator Codeloop." Codeloop. codeloop.org, June 19, 2020.https://codeloop.org/pyqt5-how-to-make-gui-calculator/.

[4] “Build a Python Desktop Calculator With PyQt.” Python GUIs. www.pythonguis.com, January 12, 2019.https://www.pythonguis.com/examples/python-calculator-gui/.

[5] “Building Calculator Using PyQt5 In Python - GeeksforGeeks.” GeeksforGeeks. www.geeksforgeeks.org, May 20,2020.https://www.geeksforgeeks.org/building-calculator-using-pyqt5-in-python/.

[6] “Basic GUI Calculator In Python - PyShark.” PyShark. pyshark.com, June 18, 2020.https://pyshark.com/basic-gui-calculator-in-python/.

[7] “Python — Simple GUI Calculator Using Tkinter - GeeksforGeeks.” GeeksforGeeks. www.geeksforgeeks.org, June 30, 2021.https://www.geeksforgeeks.org/python-simple-gui-calculator-using-tkinter/.