Laboratory Activity # 7	
Converting TUI to GUI Programs	
Course Code: CPE009B	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed:** 11/11/2024
Section: CPE21S4	**Date Submitted:** 11/11/2024
Name(s): Bona, Andrei Nycole S. Masangkay, Frederick D. Roallos, Jean Gabriel Vincent G. Santos, Andrei R. Zolina, Anna Marie L.	**Instructor:** Ma'am Maria Rizette Sayo

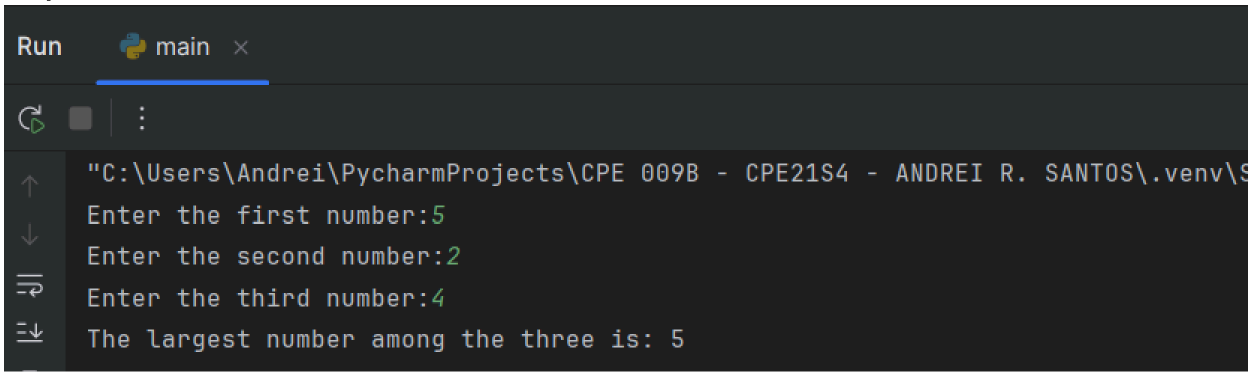
5. Procedures

Method 1

Source Code:

```python
def main():
    # Find the largest number among three numbers
    L = []
    num1 = eval(input("Enter the first number:"))
    L.append(num1)
    num2 = eval(input("Enter the second number:"))
    L.append(num2)
    num3 = eval(input("Enter the third number:"))
    L.append(num3)
    print("The largest number among the three is:", str(max(L)))

main()
```

Output:



```
"C:\Users\Andrei\PycharmProjects\CPE 009B - CPE21S4 - ANDREI R. SANTOS\.venv\
Enter the first number:5
Enter the second number:2
Enter the third number:4
The largest number among the three is: 5
```

Observation:

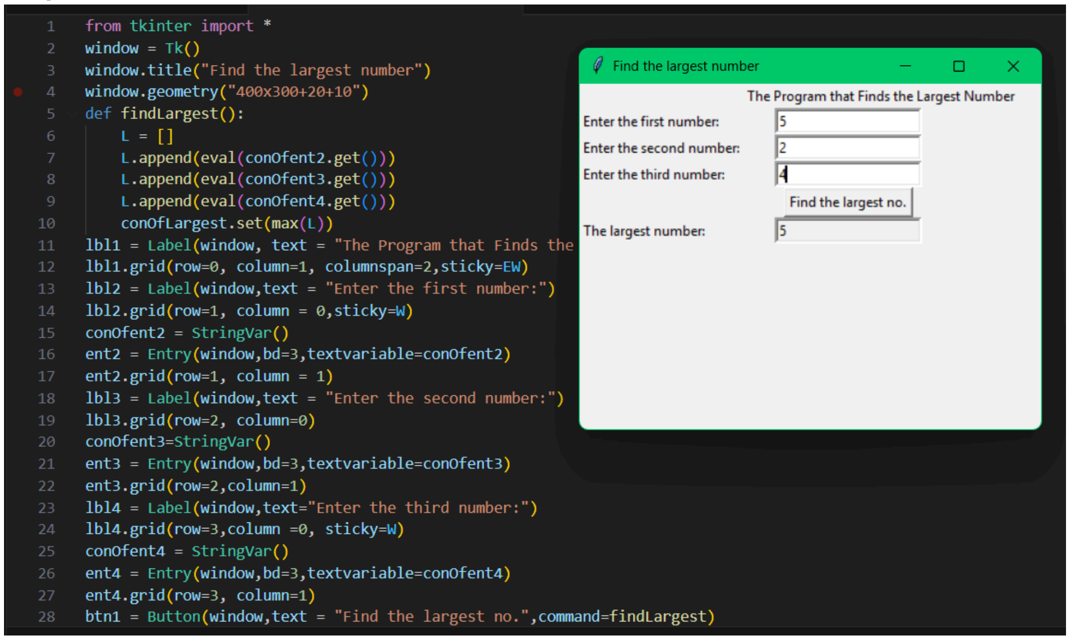
- *This program prompts the user to enter three numbers, one at a time, and then calculates the largest number using Python's'max()' function. The user enters the values into the console, and the program returns the highest number as the result. However, the use of 'eval()' indicates that the program expects valid numerical input, but it may cause problems if non-numeric or invalid input is provided.*

Method 2

Souce Code:

```python
from tkinter import *

window = Tk()
window.title("Find the largest number")
window.geometry("400x300+20+10")

def findLargest():
   L = []
   L.append(eval(conOfent2.get()))
   L.append(eval(conOfent3.get()))
   L.append(eval(conOfent4.get()))
   conOfLargest.set(max(L))

lbl1 = Label(window, text="The Program that Finds the Largest Number")
lbl1.grid(row=0, column=1, columnspan=2, sticky=EW)

lbl2 = Label(window, text="Enter the first number:")
lbl2.grid(row=1, column=0, sticky=W)

conOfent2 = StringVar()
ent2 = Entry(window, bd=3, textvariable=conOfent2)
ent2.grid(row=1, column=1)

lbl3 = Label(window, text="Enter the second number:")
lbl3.grid(row=2, column=0)

conOfent3 = StringVar()
ent3 = Entry(window, bd=3, textvariable=conOfent3)
ent3.grid(row=2, column=1)

lbl4 = Label(window, text="Enter the third number:")
lbl4.grid(row=3, column=0, sticky=W)

conOfent4 = StringVar()
ent4 = Entry(window, bd=3, textvariable=conOfent4)
ent4.grid(row=3, column=1)

btn1 = Button(window, text="Find the largest no.", command=findLargest)
btn1.grid(row=4, column=1)

lbl5 = Label(window, text="The largest number:")
lbl5.grid(row=5, column=0, sticky=W)

conOfLargest = StringVar()
ent5 = Entry(window, bd=3, state="readonly", textvariable=conOfLargest)
ent5.grid(row=5, column=1)

mainloop()
```

Output:



```
from tkinter import *
window = Tk()
window.title("Find the largest number")
window.geometry("400x300+20+10")
def findLargest():
    L = []
    L.append(eval(conOfent2.get()))
    L.append(eval(conOfent3.get()))
    L.append(eval(conOfent4.get()))
    conOfLargest.set(max(L))
lbl1 = Label(window, text = "The Program that Finds the
lbl1.grid(row=0, column=1, columnspan=2,sticky=EW)
lbl2 = Label(window,text = "Enter the first number:")
lbl2.grid(row=1, column = 0,sticky=W)
conOfent2 = StringVar()
ent2 = Entry(window,bd=3,textvariable=conOfent2)
ent2.grid(row=1, column = 1)
lbl3 = Label(window,text = "Enter the second number:")
lbl3.grid(row=2, column=0)
conOfent3=StringVar()
ent3 = Entry(window,bd=3,textvariable=conOfent3)
ent3.grid(row=2,column=1)
lbl4 = Label(window,text="Enter the third number:")
lbl4.grid(row=3,column =0, sticky=W)
conOfent4 = StringVar()
ent4 = Entry(window,bd=3,textvariable=conOfent4)
ent4.grid(row=3, column=1)
btn1 = Button(window,text = "Find the largest no.",command=findLargest)
```

```
Find the largest number
              The Program that Finds the Largest Number
Enter the first number:     5
Enter the second number:    2
Enter the third number:     4
                            Find the largest no.
The largest number:         5
```

Observation:

- *This code uses Tkinter to generate a simple GUI that allows users to enter three numbers and then select the largest one by clicking a button. It collects input via 'Entry' widgets, stores the values in a list, and calculates the maximum using Python's'max()' function. The 'eval()' function converts the string inputs from the 'Entry' widgets into numbers, which are then displayed in a read-only 'Entry' widget.*

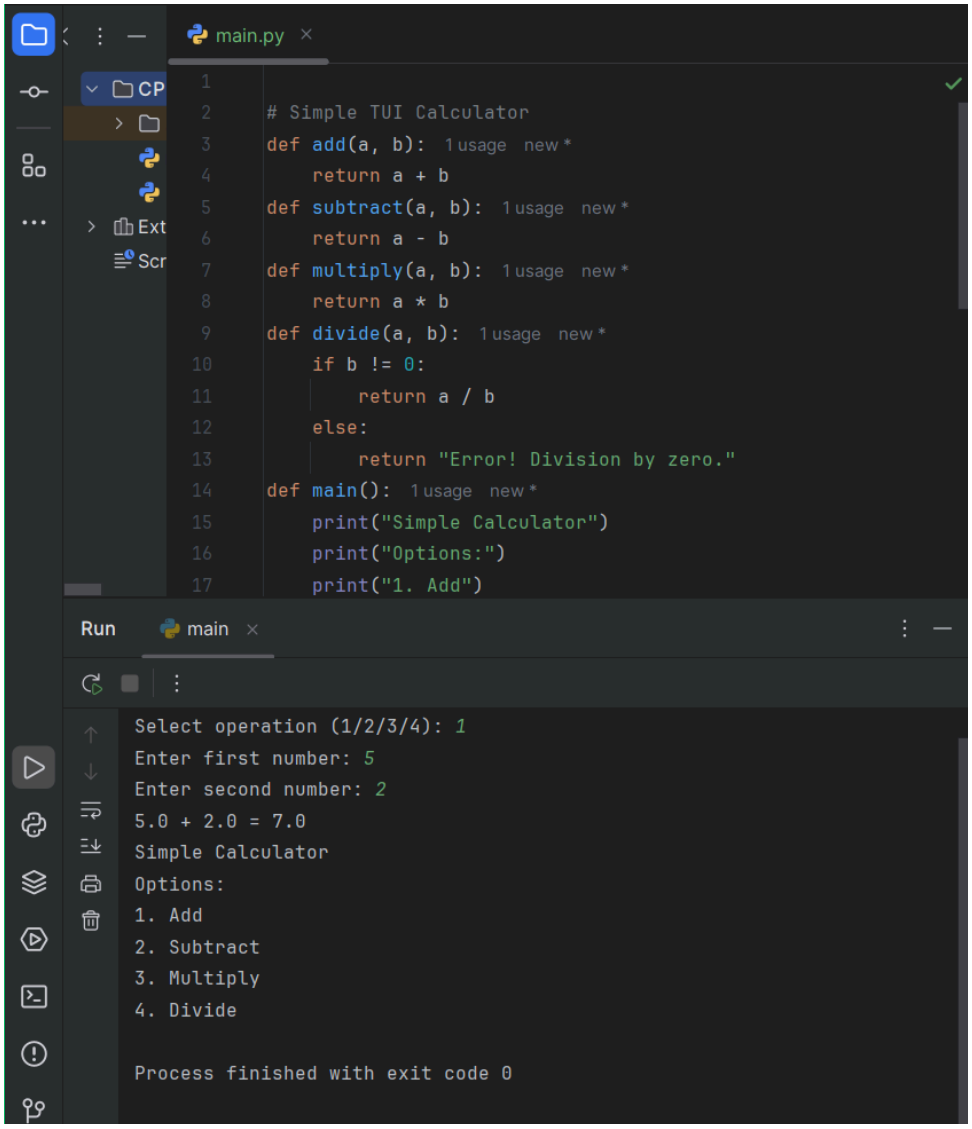
SUPPLEMENTARY ACTIVITY

#TUI IMPLEMENTATION

Source Code:

```python
# Simple TUI Calculator
def add(a, b):
  return a + b
def subtract(a, b):
  return a - b
def multiply(a, b):
  return a * b
def divide(a, b):
  if b != 0:
      return a / b
  else:
      return "Error! Division by zero."
def main():
   print("Simple Calculator")
   print("Options:")
   print("1. Add")
   print("2. Subtract")
   print("3. Multiply")
   print("4. Divide")
   choice = input("Select operation (1/2/3/4): ")
   num1 = float(input("Enter first number: "))
   num2 = float(input("Enter second number: "))
   if choice == '1':
      print(f"{num1} + {num2} = {add(num1, num2)}")
   elif choice == '2':
      print(f"{num1} - {num2} = {subtract(num1, num2)}")
   elif choice == '3':
      print(f"{num1} * {num2} = {multiply(num1, num2)}")
   elif choice == '4':
      print(f"{num1} / {num2} = {divide(num1, num2)}")
   else:
      print("Invalid input.")
if __name__ == "__main__":
  main()
```

Output:



```python
# Simple TUI Calculator
def add(a, b):
    return a + b
def subtract(a, b):
    return a - b
def multiply(a, b):
    return a * b
def divide(a, b):
    if b != 0:
        return a / b
    else:
        return "Error! Division by zero."
def main():
    print("Simple Calculator")
    print("Options:")
    print("1. Add")
```

```
Select operation (1/2/3/4): 1
Enter first number: 5
Enter second number: 2
5.0 + 2.0 = 7.0
Simple Calculator
Options:
1. Add
2. Subtract
3. Multiply
4. Divide

Process finished with exit code 0
```

Observation:

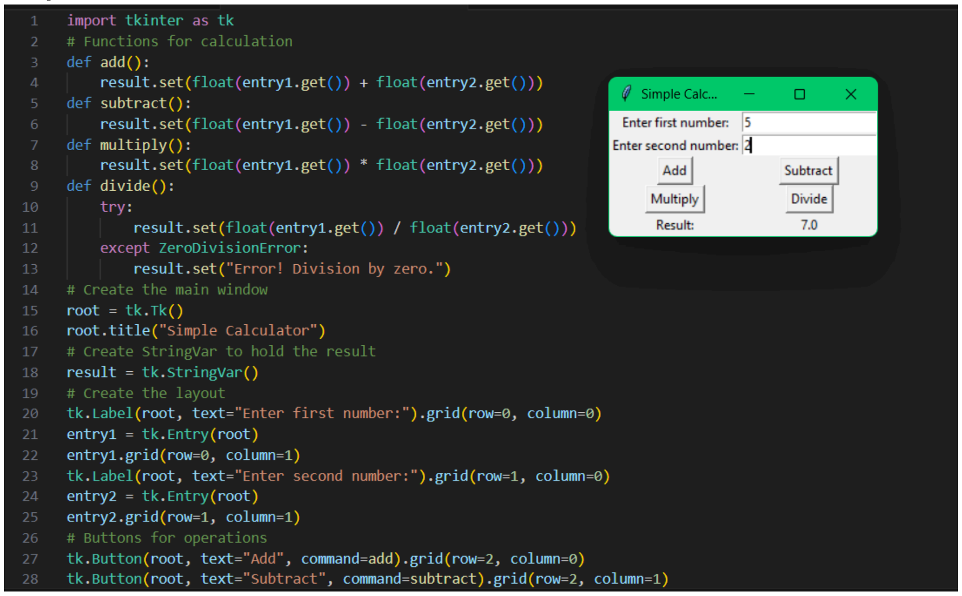
- *This is a simple text-based calculator that lets users choose an operation and enter two numbers using the command line. It executes the specified arithmetic operation and displays the result in the terminal. While functional, it lacks input validation and only supports one calculation at a time, so the program must be rerun for subsequent operations.*

#GUI IMPLEMENTATION

Source Code:

```python
import tkinter as tk
# Functions for calculation
def add():
   result.set(float(entry1.get()) + float(entry2.get()))
def subtract():
   result.set(float(entry1.get()) - float(entry2.get()))
def multiply():
    result.set(float(entry1.get()) * float(entry2.get()))
def divide():
    try:
        result.set(float(entry1.get()) / float(entry2.get()))
    except ZeroDivisionError:
        result.set("Error! Division by zero.")
# Create the main window
root = tk.Tk()
root.title("Simple Calculator")
# Create StringVar to hold the result
result = tk.StringVar()
# Create the layout
tk.Label(root, text="Enter first number:").grid(row=0, column=0)
entry1 = tk.Entry(root)
entry1.grid(row=0, column=1)
tk.Label(root, text="Enter second number:").grid(row=1, column=0)
entry2 = tk.Entry(root)
entry2.grid(row=1, column=1)
# Buttons for operations
tk.Button(root, text="Add", command=add).grid(row=2, column=0)
tk.Button(root, text="Subtract", command=subtract).grid(row=2, column=1)
tk.Button(root, text="Multiply", command=multiply).grid(row=3, column=0)
tk.Button(root, text="Divide", command=divide).grid(row=3, column=1)
# Label to show result
tk.Label(root, text="Result:").grid(row=4, column=0)
result_label = tk.Label(root, textvariable=result)
result_label.grid(row=4, column=1)
# Start the main loop
root.mainloop()
```

Output:



```python
import tkinter as tk
# Functions for calculation
def add():
    result.set(float(entry1.get()) + float(entry2.get()))
def subtract():
    result.set(float(entry1.get()) - float(entry2.get()))
def multiply():
    result.set(float(entry1.get()) * float(entry2.get()))
def divide():
    try:
        result.set(float(entry1.get()) / float(entry2.get()))
    except ZeroDivisionError:
        result.set("Error! Division by zero.")
# Create the main window
root = tk.Tk()
root.title("Simple Calculator")
# Create StringVar to hold the result
result = tk.StringVar()
# Create the layout
tk.Label(root, text="Enter first number:").grid(row=0, column=0)
entry1 = tk.Entry(root)
entry1.grid(row=0, column=1)
tk.Label(root, text="Enter second number:").grid(row=1, column=0)
entry2 = tk.Entry(root)
entry2.grid(row=1, column=1)
# Buttons for operations
tk.Button(root, text="Add", command=add).grid(row=2, column=0)
tk.Button(root, text="Subtract", command=subtract).grid(row=2, column=1)
```

```
Simple Calc...
Enter first number:  5
Enter second number: 2
Add        Subtract
Multiply   Divide
Result:    7.0
```

Observation:

- *This program uses Tkinter to create a simple calculator with a graphical user interface. Users can enter two numbers, select an operation using buttons, and see the results dynamically on the GUI. It includes error* *handling for division by zero but lacks input validation for non-numeric entries, resulting in a more interactive and user-friendly experience than the text version.*

Comparison of the Two Codes:

- *Code 3 is a simple, text-based calculator that performs one operation at a time and has no input validation, making it prone to errors. Code 4, on the other hand, includes a more interactive, graphical interface with Tkinter, allowing for continuous calculations and improved error handling for division by zero. However, neither provides proper input validation for non-numeric entries. Code 4 provides a more user-friendly experience, whereas Code 3 is simpler and lighter.*

Questions:

1. **What is TUI in Python?**

   A Terminal User Interface or TUI is a type of user interface that mostly utilizes user character inputs to interact with the program. This type of interface can be compared to the common interface used in C++ referred to as the Command Line Interface or CLI.

2. **How to make a TUI in Python?**

   A TUI-based program in Python is common when learning its basics. All IDEs that support Python as a programming language would launch a terminal for the program where the user can interact with. Programming with Python's standard library would output a TUI-based program.

3. **What is the difference between TUI and GUI?**

   The main difference of the both user interfaces lies within its level of interactivity to the user. TUI consists mostly of text, while GUI utilizes interactive elements like buttons, menus, and textboxes to communicate to its user. TUI may seem very basic to users as it is limited to only character user inputs. On the other hand, not only can GUI use keyboard inputs, but also mouse inputs that may be used for accessing buttons or menus. GUI has an advantage of being more interactive and intuitive than TUI.

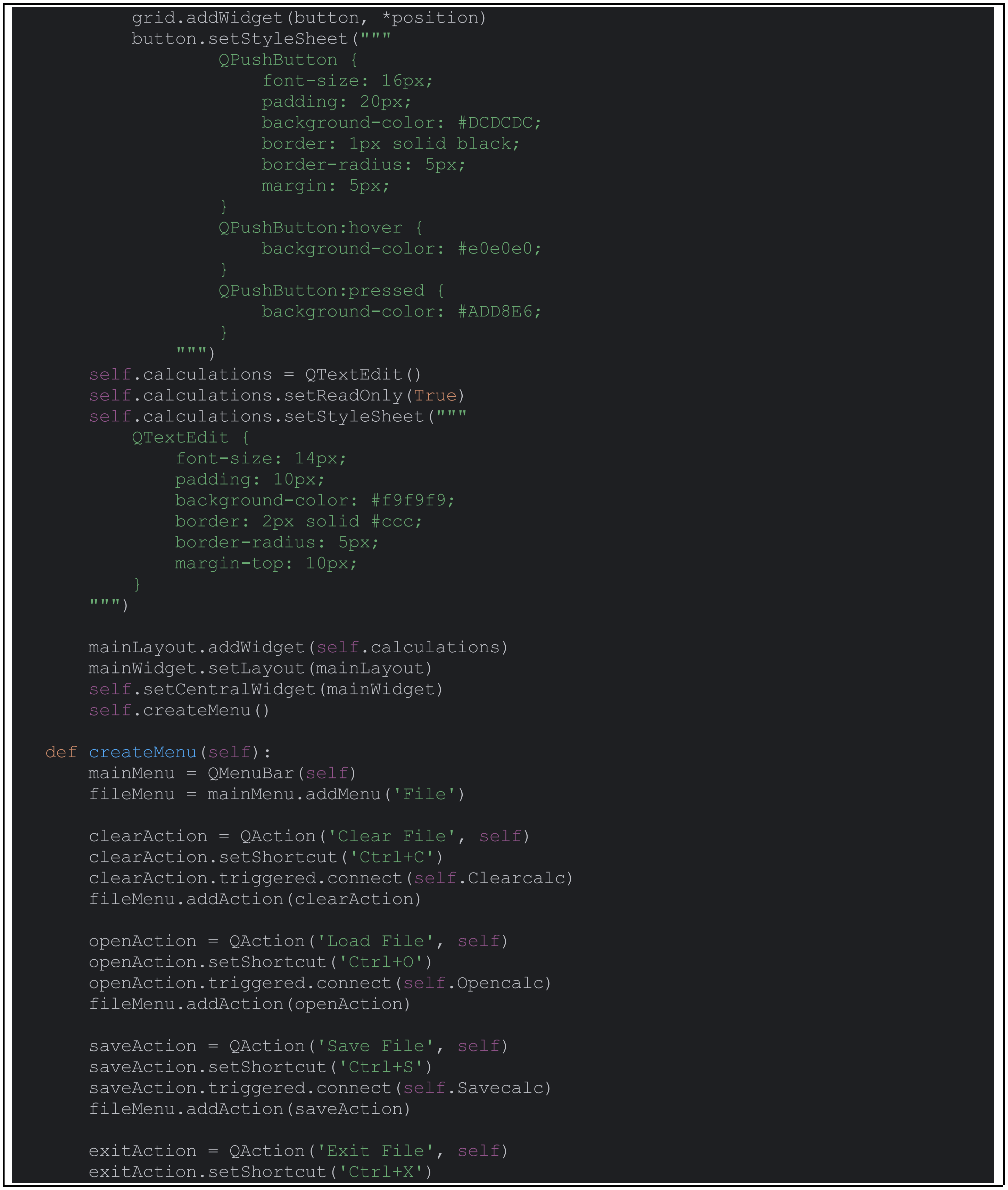
6. Supplementary Activity

1. Clear Button: Add a button to clear the input fields and reset the result.
2. History Feature: Add a list or label to show the history of operations performed.
3. Advanced Operations: Implement additional operations such as square roots, powers, or trigonometric functions.

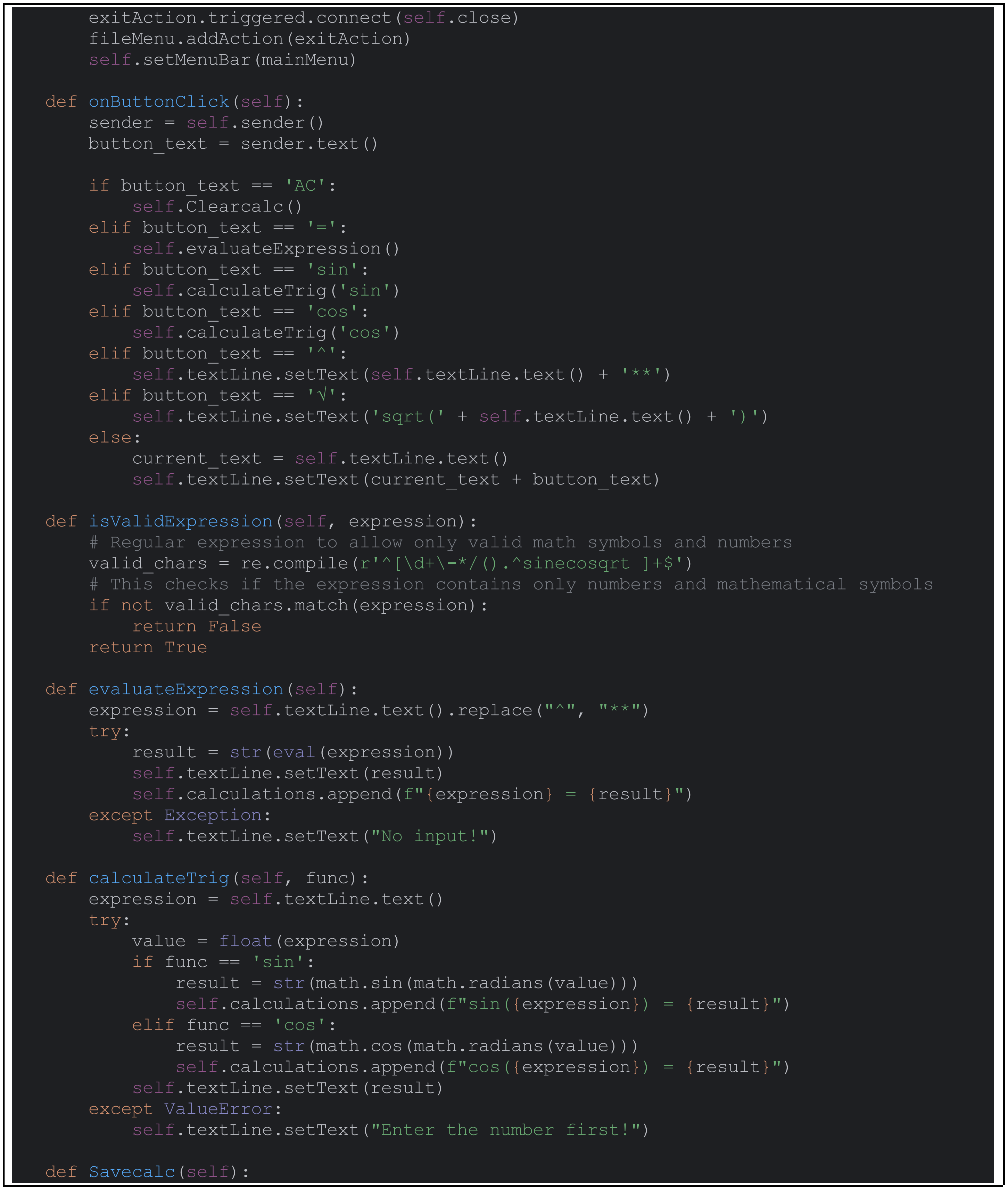
4. Input Validation: Add validation to ensure that the user only enters numeric values in the input fields.
5. Styling: Experiment with different styles (font sizes, button colors) to improve the appearance of the GUI.

Source Code(draft):

```
import sys
import math
from PyQt5.QtWidgets import QMainWindow, QWidget, QVBoxLayout, QGridLayout, QLineEdit,
QPushButton, QAction, \
   QFileDialog, QApplication, QMenuBar, QTextEdit



class Scical(QMainWindow):
   def __init__(self):
       super().__init__()
       self.setWindowTitle('~ Scientific Calculator ~')
       self.setGeometry(300, 300, 400, 400)

       self.initUI()

   def initUI(self):
       self.setStyleSheet("background-color: #FFB6C1;")

       mainWidget = QWidget(self)
       mainLayout = QVBoxLayout()
       self.textLine = QLineEdit(self)

       mainLayout.addWidget(self.textLine)
       grid = QGridLayout()
       self.textLine.setStyleSheet("""
           QLineEdit {
               font-size: 20px;
               padding: 25px;
               background-color: #CAE7D3;
               border: 2px solid black;
               border-radius: 5px;
           }
       """)
       mainLayout.addLayout(grid)

       butt = [
           '7', '8', '9', '/',
           '4', '5', '6', '*',
           '1', '2', '3', '-',
           '0', '.', '=', '+',
           'sin', 'cos', '^', 'AC',
           '√', ]

       pos = [(i, j) for i in range(6) for j in range(4)]
       for position, name in zip(pos, butt):
           button = QPushButton(name)
           button.clicked.connect(self.onButtonClick)
```



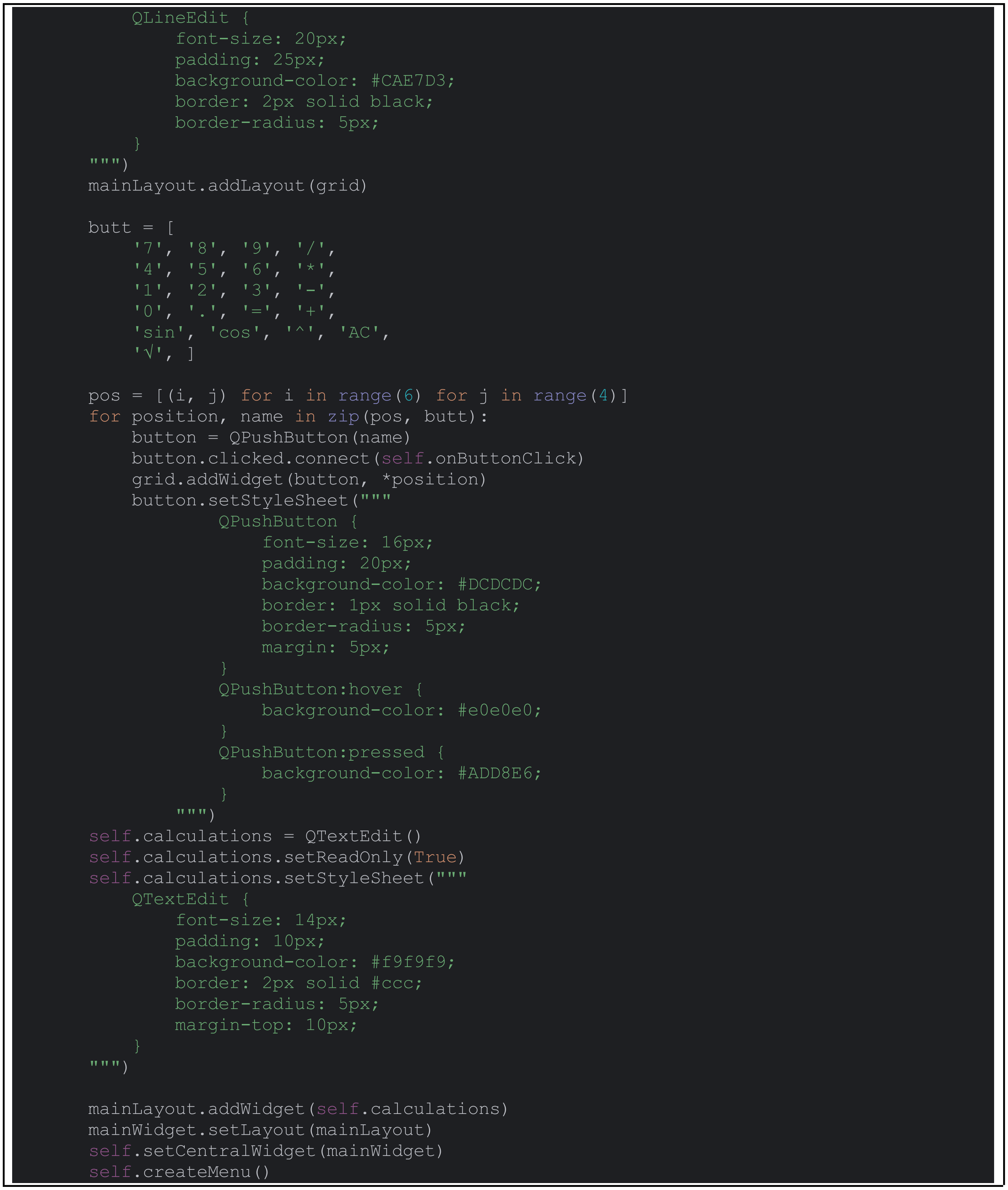
```
            grid.addWidget(button, *position)
            button.setStyleSheet("""
                    QPushButton {
                        font-size: 16px;
                        padding: 20px;
                        background-color: #DCDCDC;
                        border: 1px solid black;
                        border-radius: 5px;
                        margin: 5px;
                    }
                    QPushButton:hover {
                        background-color: #e0e0e0;
                    }
                    QPushButton:pressed {
                        background-color: #ADD8E6;
                    }
                """)
        self.calculations = QTextEdit()
        self.calculations.setReadOnly(True)
        self.calculations.setStyleSheet("""
            QTextEdit {
                font-size: 14px;
                padding: 10px;
                background-color: #f9f9f9;
                border: 2px solid #ccc;
                border-radius: 5px;
                margin-top: 10px;
            }
        """)

        mainLayout.addWidget(self.calculations)
        mainWidget.setLayout(mainLayout)
        self.setCentralWidget(mainWidget)
        self.createMenu()

    def createMenu(self):
        mainMenu = QMenuBar(self)
        fileMenu = mainMenu.addMenu('File')

        clearAction = QAction('Clear File', self)
        clearAction.setShortcut('Ctrl+C')
        clearAction.triggered.connect(self.Clearcalc)
        fileMenu.addAction(clearAction)

        openAction = QAction('Load File', self)
        openAction.setShortcut('Ctrl+O')
        openAction.triggered.connect(self.Opencalc)
        fileMenu.addAction(openAction)

        saveAction = QAction('Save File', self)
        saveAction.setShortcut('Ctrl+S')
        saveAction.triggered.connect(self.Savecalc)
        fileMenu.addAction(saveAction)

        exitAction = QAction('Exit File', self)
        exitAction.setShortcut('Ctrl+X')
```



```
        exitAction.triggered.connect(self.close)
        fileMenu.addAction(exitAction)
        self.setMenuBar(mainMenu)

    def onButtonClick(self):
        sender = self.sender()
        button_text = sender.text()

        if button_text == 'AC':
            self.Clearcalc()
        elif button_text == '=':
            self.evaluateExpression()
        elif button_text == 'sin':
            self.calculateTrig('sin')
        elif button_text == 'cos':
            self.calculateTrig('cos')
        elif button_text == '^':
            self.textLine.setText(self.textLine.text() + '**')
        elif button_text == '√':
            self.textLine.setText('sqrt(' + self.textLine.text() + ')')
        else:
            current_text = self.textLine.text()
            self.textLine.setText(current_text + button_text)

    def isValidExpression(self, expression):
        # Regular expression to allow only valid math symbols and numbers
        valid_chars = re.compile(r'^[\d+\-*/().^sinecosqrt ]+$')
        # This checks if the expression contains only numbers and mathematical symbols
        if not valid_chars.match(expression):
            return False
        return True

    def evaluateExpression(self):
        expression = self.textLine.text().replace("^", "**")
        try:
            result = str(eval(expression))
            self.textLine.setText(result)
            self.calculations.append(f"{expression} = {result}")
        except Exception:
            self.textLine.setText("No input!")

    def calculateTrig(self, func):
        expression = self.textLine.text()
        try:
            value = float(expression)
            if func == 'sin':
                result = str(math.sin(math.radians(value)))
                self.calculations.append(f"sin({expression}) = {result}")
            elif func == 'cos':
                result = str(math.cos(math.radians(value)))
                self.calculations.append(f"cos({expression}) = {result}")
            self.textLine.setText(result)
        except ValueError:
            self.textLine.setText("Enter the number first!")

    def Savecalc(self):
       options = QFileDialog.Options()
       fileName, _ = QFileDialog.getSaveFileName(self, "Save File", "", "Text Files
(*.txt)", options=options)
       if fileName:
           with open(fileName, 'w') as file:
               file.write(self.calculations.toPlainText())

   def Opencalc(self):
       options = QFileDialog.Options()
       fileName, _ = QFileDialog.getOpenFileName(self, "Load File", "", "Text Files
(*.txt)", options=options)
       if fileName:
           with open(fileName, 'r') as file:
               data = file.read()
               self.calculations.setText(data)

   def Clearcalc(self):
       self.textLine.clear()


if __name__ == '__main__':
   app = QApplication(sys.argv)
   calculator = Scical()
   calculator.show()
   sys.exit(app.exec_())
```

Another Source Code:

```
import sys
import math
import re
from PyQt5.QtWidgets import QMainWindow, QWidget, QVBoxLayout, QGridLayout, QLineEdit,
QPushButton, QAction, \
   QFileDialog, QApplication, QMenuBar, QTextEdit


class Scical(QMainWindow):
   def __init__(self):
       super().__init__()
       self.setWindowTitle('~ Scientific Calculator ~')
       self.setGeometry(300, 300, 400, 400)

       self.initUI()

   def initUI(self):
       self.setStyleSheet("background-color: #FFB6C1;")

       mainWidget = QWidget(self)
       mainLayout = QVBoxLayout()
       self.textLine = QLineEdit(self)

       mainLayout.addWidget(self.textLine)
       grid = QGridLayout()
       self.textLine.setStyleSheet("""
```



```python
            QLineEdit {
                font-size: 20px;
                padding: 25px;
                background-color: #CAE7D3;
                border: 2px solid black;
                border-radius: 5px;
            }
        """)
        mainLayout.addLayout(grid)

        butt = [
            '7', '8', '9', '/',
            '4', '5', '6', '*',
            '1', '2', '3', '-',
            '0', '.', '=', '+',
            'sin', 'cos', '^', 'AC',
            '√', ]

        pos = [(i, j) for i in range(6) for j in range(4)]
        for position, name in zip(pos, butt):
            button = QPushButton(name)
            button.clicked.connect(self.onButtonClick)
            grid.addWidget(button, *position)
            button.setStyleSheet("""
                    QPushButton {
                        font-size: 16px;
                        padding: 20px;
                        background-color: #DCDCDC;
                        border: 1px solid black;
                        border-radius: 5px;
                        margin: 5px;
                    }
                    QPushButton:hover {
                        background-color: #e0e0e0;
                    }
                    QPushButton:pressed {
                        background-color: #ADD8E6;
                    }
                """)
        self.calculations = QTextEdit()
        self.calculations.setReadOnly(True)
        self.calculations.setStyleSheet("""
            QTextEdit {
                font-size: 14px;
                padding: 10px;
                background-color: #f9f9f9;
                border: 2px solid #ccc;
                border-radius: 5px;
                margin-top: 10px;
            }
        """)

        mainLayout.addWidget(self.calculations)
        mainWidget.setLayout(mainLayout)
        self.setCentralWidget(mainWidget)
        self.createMenu()
```

```
    def createMenu(self):
        mainMenu = QMenuBar(self)
        fileMenu = mainMenu.addMenu('File')

        clearAction = QAction('Clear File', self)
        clearAction.setShortcut('Ctrl+C')
        clearAction.triggered.connect(self.Clearcalc)
        fileMenu.addAction(clearAction)

        openAction = QAction('Load File', self)
        openAction.setShortcut('Ctrl+O')
        openAction.triggered.connect(self.Opencalc)
        fileMenu.addAction(openAction)

        saveAction = QAction('Save File', self)
        saveAction.setShortcut('Ctrl+S')
        saveAction.triggered.connect(self.Savecalc)
        fileMenu.addAction(saveAction)

        exitAction = QAction('Exit File', self)
        exitAction.setShortcut('Ctrl+X')
        exitAction.triggered.connect(self.close)
        fileMenu.addAction(exitAction)
        self.setMenuBar(mainMenu)

    def onButtonClick(self):
        sender = self.sender()
        button_text = sender.text()

        if button_text == 'AC':
            self.Clearcalc()
        elif button_text == '=':
            self.evaluateExpression()
        elif button_text == 'sin':
            self.calculateTrig('sin')
        elif button_text == 'cos':
            self.calculateTrig('cos')
        elif button_text == '^':
            self.textLine.setText(self.textLine.text() + '**')
        elif button_text == '√':
            # Fix for square root button
            current_text = self.textLine.text()
            self.textLine.setText(f'math.sqrt({current_text})')  # Replace √ with
math.sqrt()
        else:
            current_text = self.textLine.text()
            self.textLine.setText(current_text + button_text)

    def isValidExpression(self, expression):
        # Regular expression to allow only valid math symbols and numbers
        valid_chars = re.compile(r'^[\d+\-*/().^sinecosqrt ]+$')
        # This checks if the expression contains only numbers and mathematical symbols
        if not valid_chars.match(expression):
            return False
        return True
```

```
   def evaluateExpression(self):
       expression = self.textLine.text().replace("^", "**")
       try:
           # Evaluate the expression and display the result
           result = str(eval(expression))
           self.textLine.setText(result)
           self.calculations.append(f"{expression} = {result}")
       except Exception:
           self.textLine.setText("No input!")

   def calculateTrig(self, func):
       expression = self.textLine.text()
       try:
           value = float(expression)
           if func == 'sin':
               result = str(math.sin(math.radians(value)))
               self.calculations.append(f"sin({expression}) = {result}")
           elif func == 'cos':
               result = str(math.cos(math.radians(value)))
               self.calculations.append(f"cos({expression}) = {result}")
           self.textLine.setText(result)
       except ValueError:
           self.textLine.setText("Enter the number first!")

   def Savecalc(self):
       options = QFileDialog.Options()
       fileName, _ = QFileDialog.getSaveFileName(self, "Save File", "", "Text Files
(*.txt)", options=options)
       if fileName:
           with open(fileName, 'w') as file:
               file.write(self.calculations.toPlainText())

   def Opencalc(self):
       options = QFileDialog.Options()
       fileName, _ = QFileDialog.getOpenFileName(self, "Load File", "", "Text Files
(*.txt)", options=options)
       if fileName:
           with open(fileName, 'r') as file:
               data = file.read()
               self.calculations.setText(data)

   def Clearcalc(self):
       self.textLine.clear()


if __name__ == '__main__':
   app = QApplication(sys.argv)
   calculator = Scical()
   calculator.show()
   sys.exit(app.exec_())
```

7. Conclusion

In conclusion, a Text User Interface or TUI is ideal for command-based applications, while a Graphical User Interface or GUI is flexible generally more user-friendly, and appropriate for wider applications. Making use of the advantageous capabilities of GUIs will finally make it possible for us to create interactive, graphically enhanced, and user-friendly software.

Furthermore, TUI and GUI are what need to be taken in designing software applications for various user needs and environments. A TUI is based on typed text commands that are indeed fast and efficient for experienced users who want direct access to the system. In most cases, it is applied in server management and programming. On the other hand, GUI uses visual icons such as buttons and windows for people to access the application. Most consumer software, from operating systems to mobile apps, makes use of GUIs because they are very easy to navigate.

The calculator created in the supplementary activity is a very good example of an application of GUI. Adding such features as the Clear button, history tracking, advanced functions like square roots and trigonometric functions, input validation, and custom styles makes the calculator easy to use and easy to navigate. The GUI helps make it easy for the user to perform complex operations without memorizing the commands.

9. Assessment Rubric