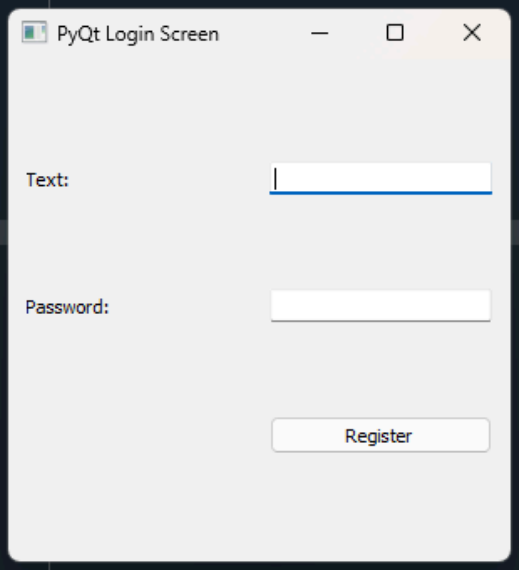
Activity 6	
GUI Design: Layout and Styling	
Course Code: CPE 009B	**Program**: Computer Engineering
Course Title: Object Oriented Programming 2	**Date Performed**: 10/28/24
Section: CPE21S4	**Date Submitted**: 10/28/24
Name: Dominic Joseph P. Virtucio	**Instructor**: Ma'am Sayo

Procedure and Output

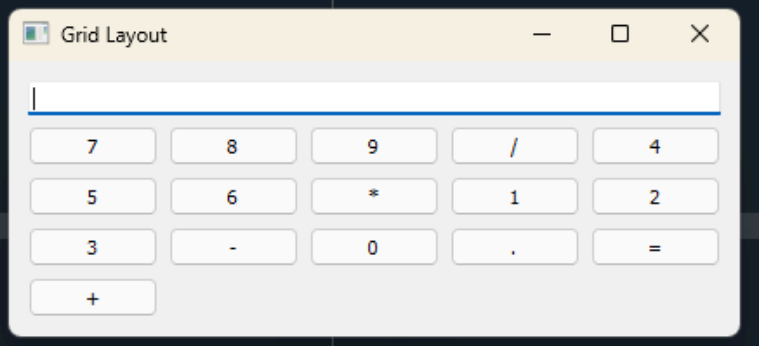
Basic Grid Layout

```
import sys
from PyQt5.QtWidgets import QMainWindow, QApplication, QWidget, QGridLayout, QLabel, QLineEdit, QPushButton
from PyQt5.QtGui import QIcon

class App(QWidget):
    def __init__(self):
        super().__init__()
        self.title = "PyQt Login Screen"
        self.x = 200  # or left
        self.y = 200  # or top
        self.width = 300
        self.height = 300
        self.initUI()

    def initUI(self):
        self.setWindowTitle(self.title)
        self.setGeometry(self.x, self.y, self.width, self.height)
        self.setWindowIcon(QIcon('pythonico.ico'))
        self.createGridLayout()
        self.setLayout(self.layout)
        self.show()

    def createGridLayout(self):
        self.layout = QGridLayout()
        self.layout.setColumnStretch(1, 2)
        self.textboxlbl = QLabel("Text: ", self)
        self.textbox = QLineEdit(self)
        self.passwordlbl = QLabel("Password: ", self)
        self.password = QLineEdit(self)
        self.password.setEchoMode(QLineEdit.Password)
        self.button = QPushButton('Register', self)
        self.button.setToolTip("You've hovered over me!")
        self.layout.addWidget(self.textboxlbl, 0, 1)
        self.layout.addWidget(self.textbox, 0, 2)
        self.layout.addWidget(self.passwordlbl, 1, 1)
        self.layout.addWidget(self.password, 1, 2)
        self.layout.addWidget(self.button, 2, 2)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = App()
    sys.exit(app.exec_())
```

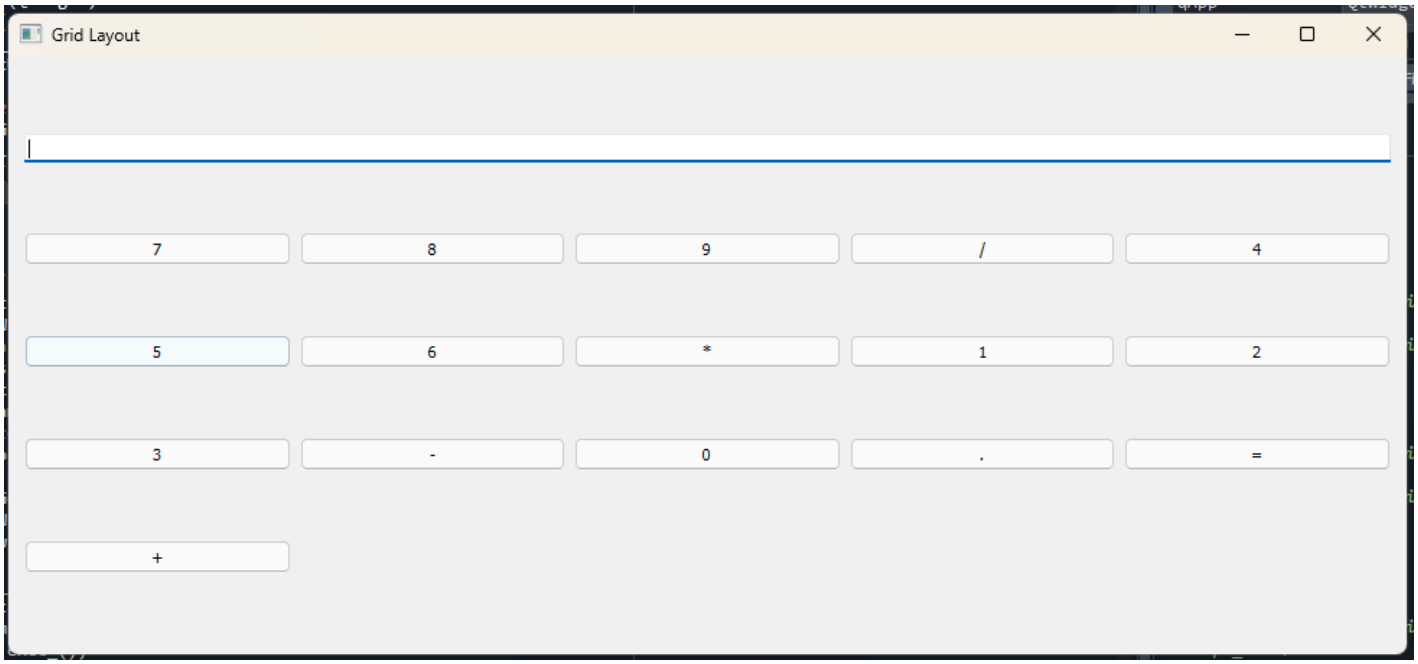


Grid Layout using Loops

```python
#Grid Layout
import sys
from PyQt5.QtWidgets import QGridLayout, QLineEdit, QPushButton, QHBoxLayout, QVBoxLayout, QWidget, QApplication

class GridExample(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        grid = QGridLayout()
        self.setLayout(grid)
        names = [
            '7', '8', '9', '/', ''
            '4', '5', '6', '*', ''
            '1', '2', '3', '-', ''
            '0', '.', '=', '+', ''
            '', '', '', '', '']
        self.textLine = QLineEdit(self)
        grid.addWidget(self.textLine, 0, 1, 1, 5)
        # using a loop to generate positions
        positions = [(i, j) for i in range(1, 7) for j in range(1, 6)]
        for position, name in zip(positions, names):
            if name == '':
                continue
            button = QPushButton(name)
            grid.addWidget(button, *position)
        self.setGeometry(300, 300, 300, 150)
        self.setWindowTitle('Grid Layout')
        self.show()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = GridExample()
    sys.exit(app.exec_())
```



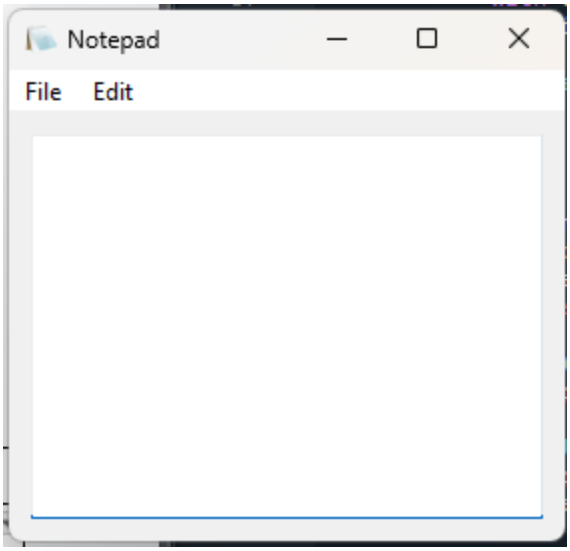
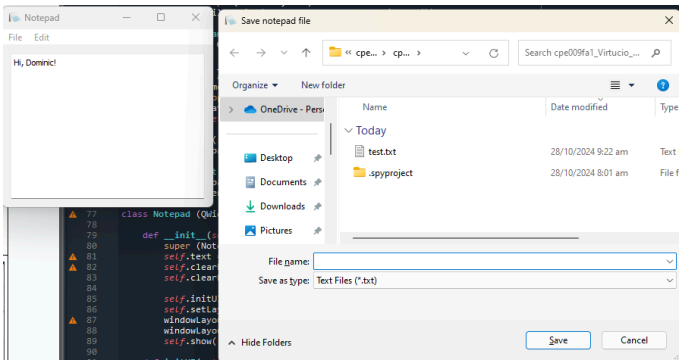
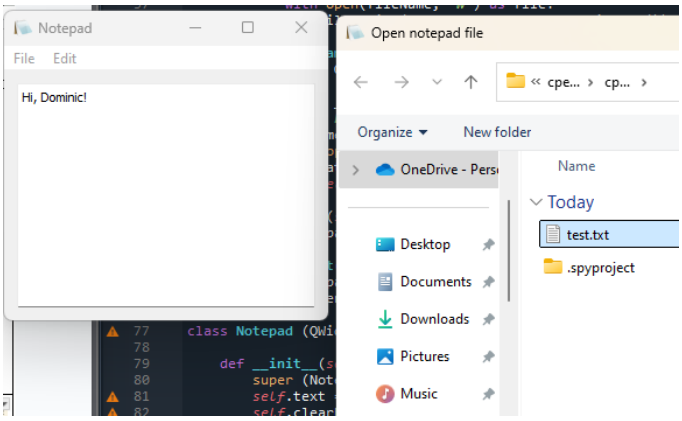
Stretched:

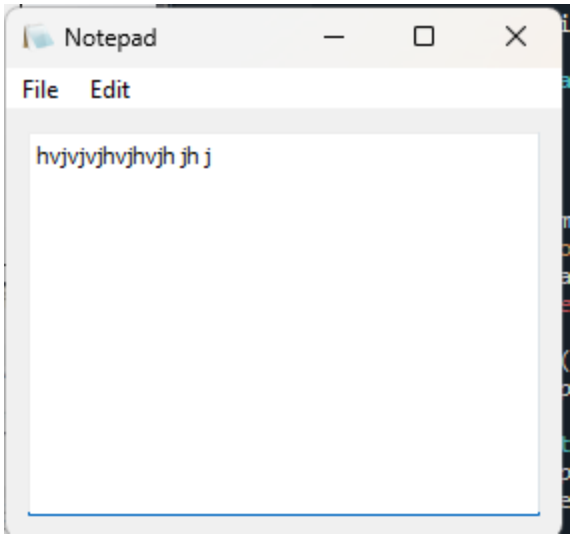
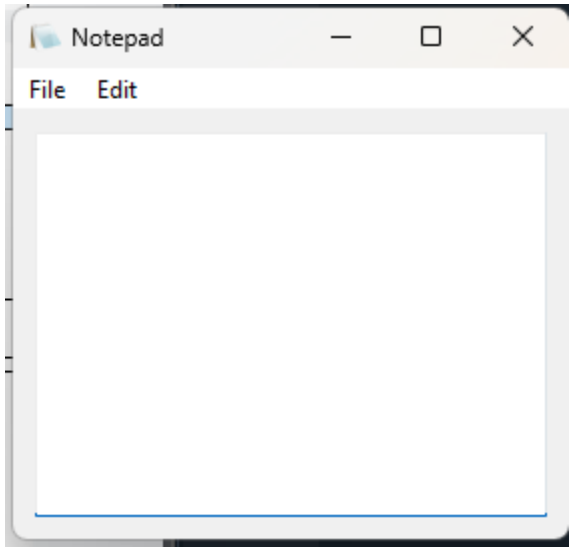
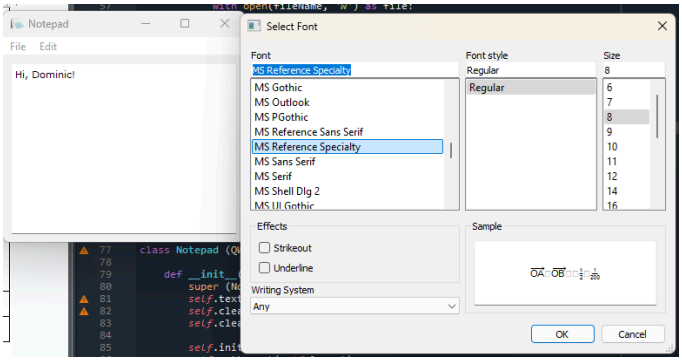


Vbox and Hbox layout managers (Simple Notepad)

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import QIcon

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Notepad")
        self.setWindowIcon(QIcon('pythonico.ico'))
        self.loadmenu()

        self.loadwidget()
        self.show()

    def loadmenu(self):
        mainMenu = self.menuBar()
        fileMenu = mainMenu.addMenu('File')
        editMenu = mainMenu.addMenu('Edit')

        editButton = QAction('Clear', self)
        editButton.setShortcut('ctrl+M')
        editButton.triggered.connect(self.cleartext)
        editMenu.addAction(editButton)

        fontButton = QAction('Font', self)
        fontButton.setShortcut('ctrl+D')
        fontButton.triggered.connect(self.showFontDialog)
        editMenu.addAction(fontButton)

        saveButton = QAction('Save', self)
        saveButton.setShortcut('Ctrl+S')
        saveButton.triggered.connect(self.saveFileDialog)
        fileMenu.addAction(saveButton)

        openButton = QAction('Open', self)
        openButton.setShortcut('Ctrl+O')
        openButton.triggered.connect(self.openFileNameDialog)
        fileMenu.addAction(openButton)

        exitButton = QAction('Exit', self)
        exitButton.setShortcut('Ctrl+Q')
        exitButton.setStatusTip('Exit application')
        exitButton.triggered.connect(self.close)
        fileMenu.addAction(exitButton)

    def showFontDialog(self):
        font, ok = QFontDialog.getFont()
        if ok:
            self.notepad.text.setFont(font)
```

```
    def saveFileDialog(self):
        options = QFileDialog.Options()
        # options |= QFileDialog.DontUseNativeDialog
        fileName, _ = QFileDialog.getSaveFileName(self, "Save notepad file", "",
            "Text Files (*.txt);;Python Files (*.py);;All files (*)", options=options)
        if fileName:
            with open(fileName, 'w') as file:
                file.write(self.notepad.text.toPlainText())

    def openFileNameDialog(self):
        options = QFileDialog.Options()
        # options |= QFileDialog.DontUseNativeDialog
        fileName, _ = QFileDialog.getOpenFileName(self, "Open notepad file", "",
            "Text Files (*.txt);;Python Files (*.py);;All files (*)", options=options)
        if fileName:
            with open(fileName, 'r') as file:
                data = file.read()
                self.notepad.text.setText(data)

    def cleartext(self):
        self.notepad.text.clear()

    def loadwidget(self):
        self.notepad = Notepad()
        self.setCentralWidget(self.notepad)

class Notepad (QWidget):

    def __init__(self):
        super (Notepad, self).__init__()
        self.text = QTextEdit(self)
        self.clearbtn= QPushButton("Clear")
        self.clearbtn.clicked.connect(self.cleartext)

        self.initUI()
        self.setLayout(self.layout)
        windowLayout = QVBoxLayout()
        windowLayout.addWidget(self.horizontalGroupBox)
        self.show()

    def initUI(self):
        self.horizontalGroupBox = QGroupBox("Grid")
        self.layout = QHBoxLayout()
        self.layout.addWidget(self.text)
        # self.layout.addWidget(self.clearbtn)
        self.horizontalGroupBox.setLayout(self.layout)

    def cleartext(self):Z
        self.text.clear()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = MainWindow()
    sys.exit(app.exec_())
```

Output	
Base output	
Save Option (Ctrl + S)	
Open File (Ctrl + O)	
Exit (Ctrl + Q)	***window disappears****

Clear (Ctrl + M) **Before:** **After:**	 
Font (Ctrl + D)	

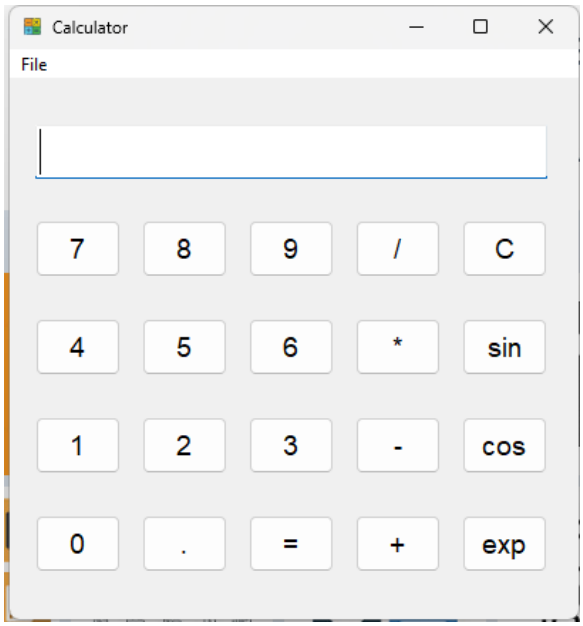
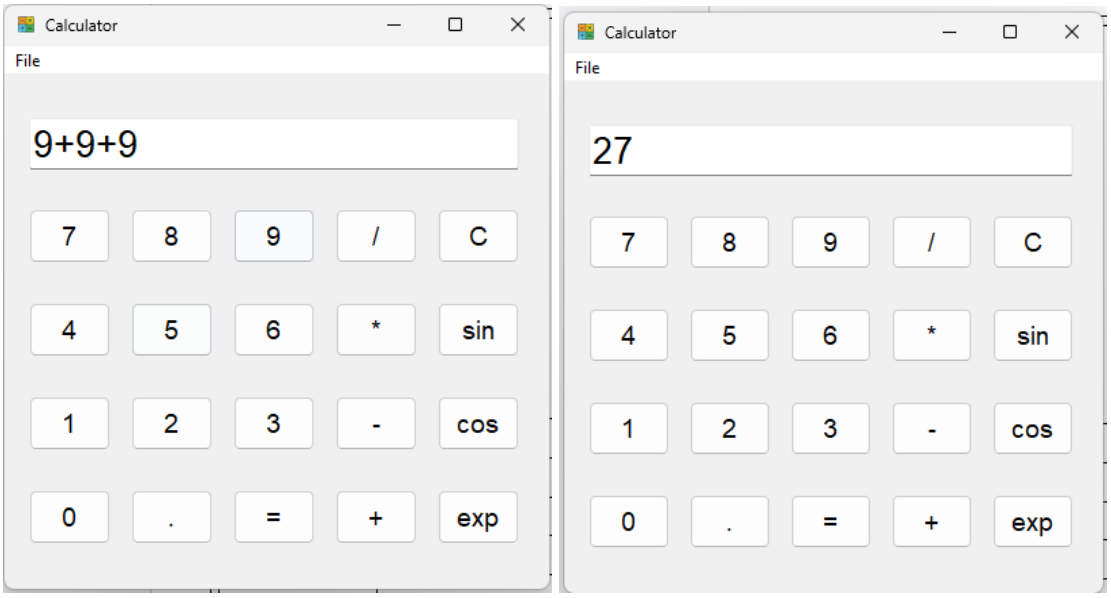
Supplementary Activity

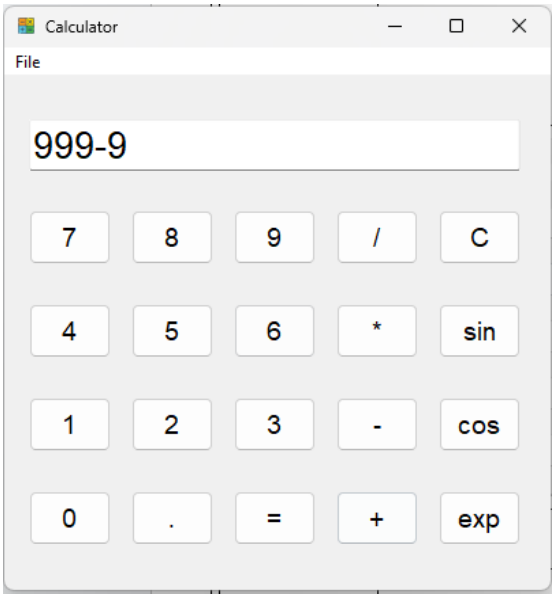
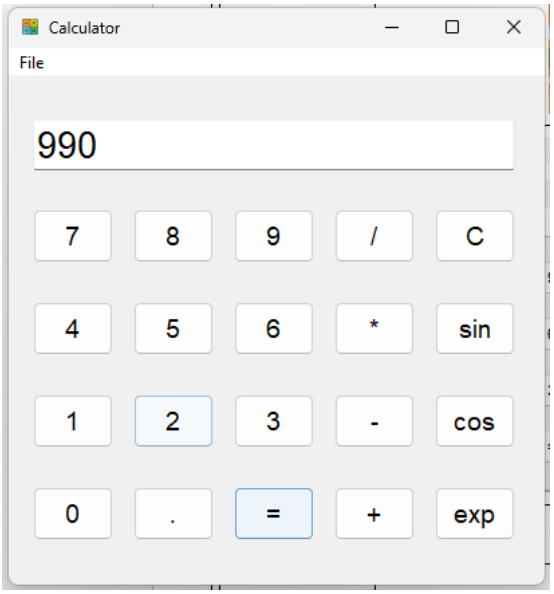
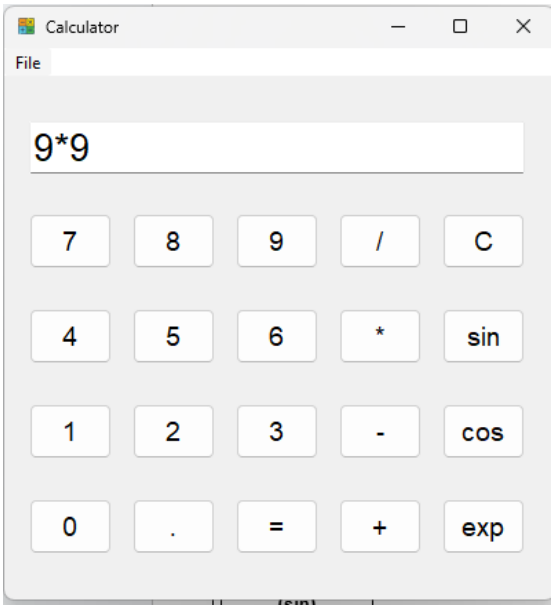
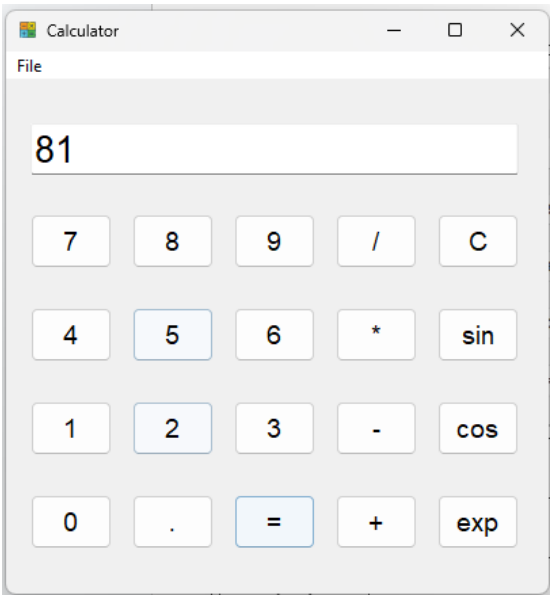
```python
import sys
import math
```

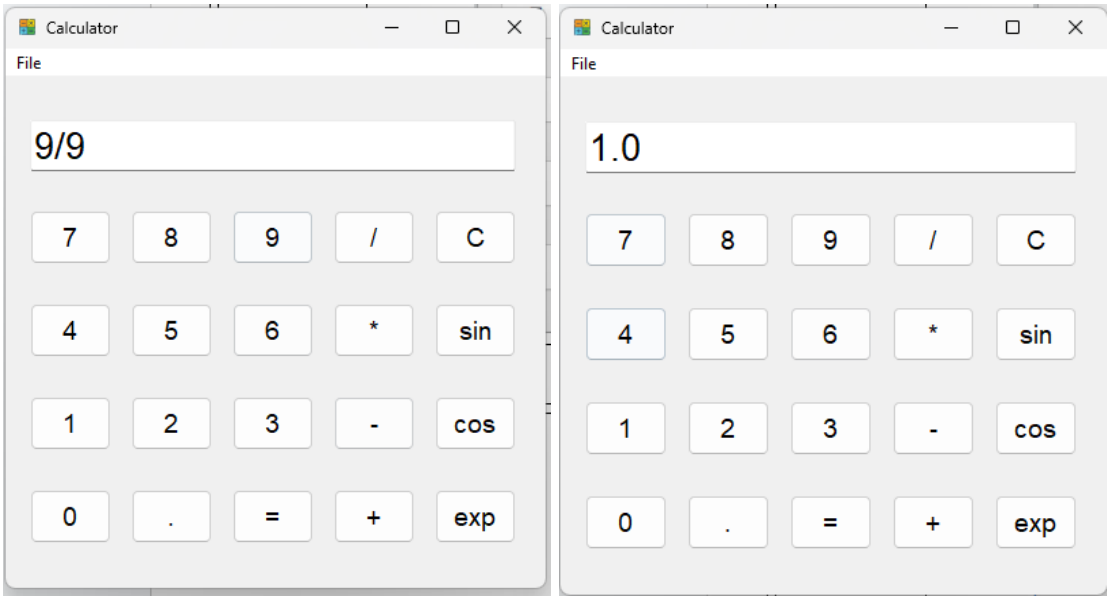
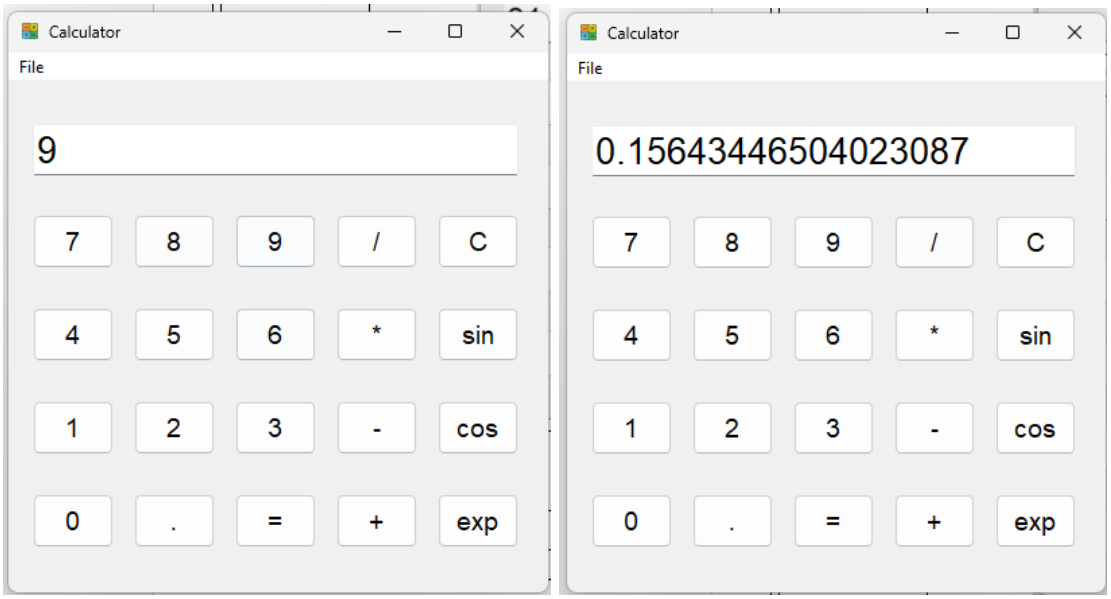
```
from PyQt5.QtWidgets import (
    QMainWindow, QApplication, QWidget, QGridLayout, QLineEdit,
    QPushButton, QAction, QMessageBox
)
from PyQt5.QtGui import QFont, QIcon
from PyQt5.QtCore import QSize

class Calculator(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Calculator")
        self.setWindowIcon(QIcon('pythonico.ico'))
        self.setGeometry(300, 300, 400, 400)
        self.initUI()
        self.history_file = "calculator_history.txt"

    def initUI(self):
        self.central_widget = QWidget(self)
        self.setCentralWidget(self.central_widget)

        self.grid = QGridLayout(self.central_widget)
        self.central_widget.setLayout(self.grid)

        self.textLine = QLineEdit(self)
        self.textLine.setReadOnly(False)
        self.textLine.setFont(QFont('Arial', 20))
        self.grid.addWidget(self.textLine, 0, 0, 1, 0)

        names = [
            '7', '8', '9', '/', 'C',
            '4', '5', '6', '*', 'sin',
            '1', '2', '3', '-', 'cos',
            '0', '.', '=', '+', 'exp'
        ]

        positions = [(i, j) for i in range(1, 6) for j in range(5)]
        for position, name in zip(positions, names):
            button = QPushButton(name)
            button.setFont(QFont('Arial', 14))
            button.setFixedSize(QSize(60, 40))
            button.clicked.connect(self.on_button_clicked)
            self.grid.addWidget(button, *position)

        self.load_menu()

    def load_menu(self):
        mainMenu = self.menuBar()
        fileMenu = mainMenu.addMenu('File')

        clear_history_action = QAction('Clear History', self)
        clear_history_action.triggered.connect(self.clear_history)
        fileMenu.addAction(clear_history_action)

        exit_action = QAction('Exit', self)
        exit_action.triggered.connect(self.close)
        exit_action.setShortcut('Ctrl+Q')  # Set shortcut for exiting
        fileMenu.addAction(exit_action)

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

        if button_text == 'C':
```

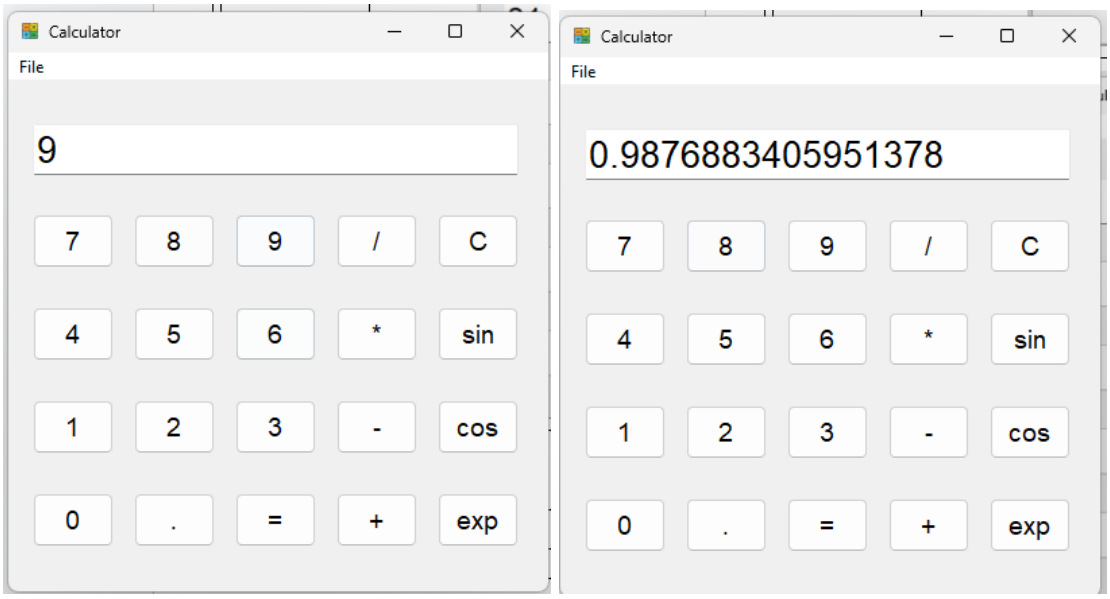
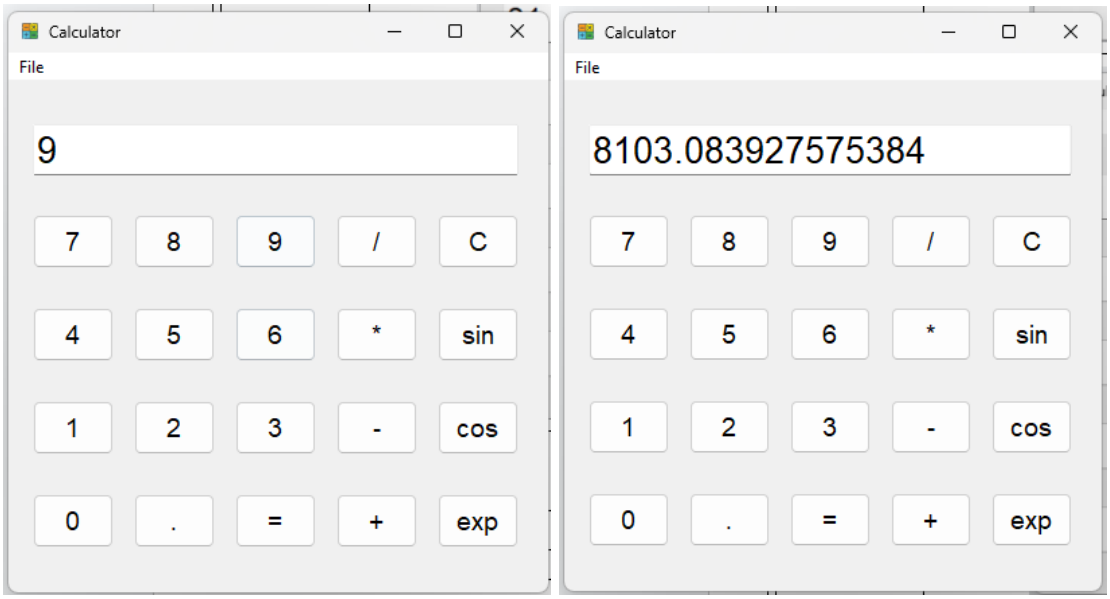
```
            self.textLine.clear()
        elif button_text == '=':
            self.calculate_result()
        elif button_text in ['sin', 'cos', 'exp']:
            self.perform_trig_or_exp(button_text)
        else:
            current_text = self.textLine.text()
            new_text = current_text + button_text
            self.textLine.setText(new_text)

    def calculate_result(self):
        expression = self.textLine.text()
        try:
            result = eval(expression)
            self.textLine.setText(str(result))
            self.save_to_history(f"{expression} = {result}")
        except Exception as e:
            QMessageBox.critical(self, "Error", f"Invalid Expression: {str(e)}")

    def perform_trig_or_exp(self, operation):
        try:
            value = float(self.textLine.text())
            if operation == 'sin':
                result = math.sin(math.radians(value))
            elif operation == 'cos':
                result = math.cos(math.radians(value))
            elif operation == 'exp':
                result = math.exp(value)
            self.textLine.setText(str(result))
            self.save_to_history(f"{operation}({value}) = {result}")
        except ValueError:
            QMessageBox.critical(self, "Error", "Please enter a valid number.")
        except Exception as e:
            QMessageBox.critical(self, "Error", str(e))

    def save_to_history(self, entry):
        with open(self.history_file, 'a') as f:
            f.write(entry + '\n')

    def clear_history(self):
        try:
            with open(self.history_file, 'w') as f:
                f.truncate()
            QMessageBox.information(self, "Success", "History cleared.")
        except Exception as e:
            QMessageBox.critical(self, "Error", str(e))

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

Output:

Base Output	
Add Operation (+)	

Subtract Operation (-)	 
Multiply Operation(*)	 

Divide Operation(/)	
Sine Operation (sin)	

Cosine Operation (cos)	
Exponential Operation (exp)	

Conclusion:
In conclusion, my comprehension of graphical user interface programming has been greatly improved by this lab exercise on GUI Design: Layout and Styling. I had practical experience with important GUI components by developing programs such as a calculator and a basic notepad. I became proficient with layout managers like GridLayout, VBox, and HBox, which enabled me to arrange components in a responsive and organized manner. My applications now effortlessly adjust to different window sizes because of my knowledge of dynamic widget positioning and sizing, which improves user experience.

In addition to layout management, I looked into event handling and widget customization. I successfully implemented button click events and keyboard shortcuts, resulting in interactive interfaces with immediate feedback. Customizing GUI elements with different fonts, sizes, and icons enabled me to improve the visual appeal of my applications. Furthermore, I included practical functionalities such as file operations and mathematical calculations to show how effective layout design and functionality can coexist. This hands-on experience has provided me with valuable skills in GUI development with PyQt5, which I hope to apply to future software projects that require intuitive and user-friendly interfaces.