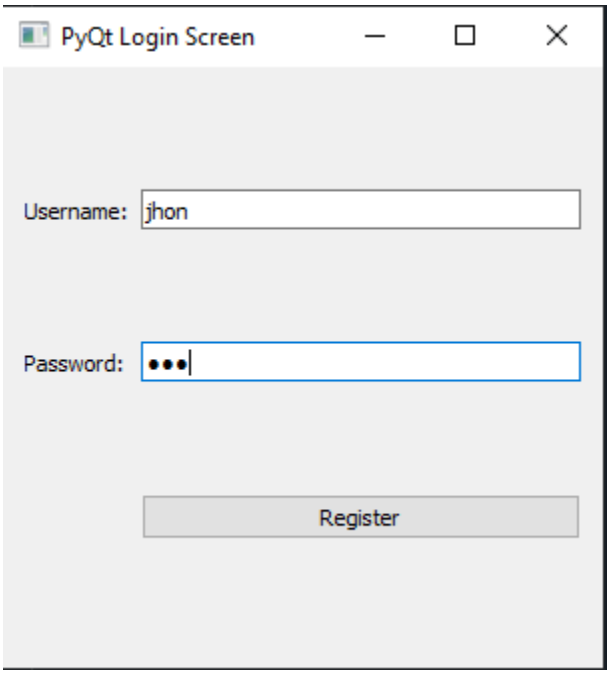
Laboratory Activity 6 - GUI Design: Layout and Styling	
Bautista, Jhon Hendricks T.	11/06/2024
Course/Section: CPE009B/CPE21S1	Mrs. Maria Rizette Sayo

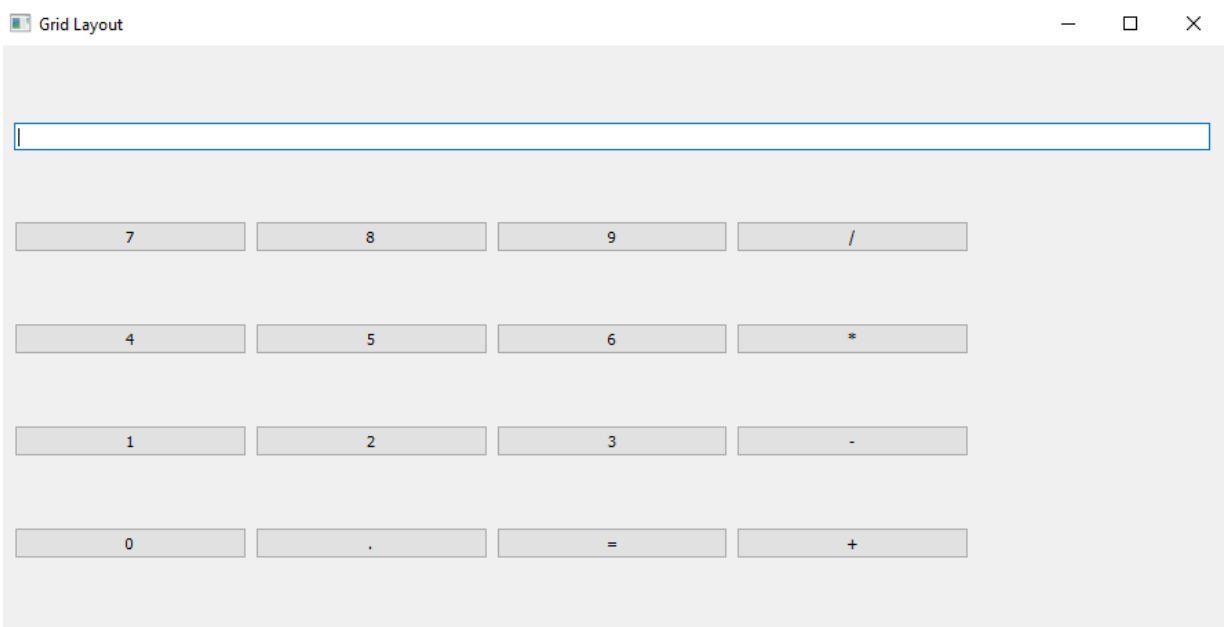
PROCEDURE



Grid Layout

```python
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)

        # Button names for the calculator
        names = [
            '7', '8', '9', '/', '4',
            '5', '6', '*', '1', '2',
            '3', '-', '0', '.', '=', '+'
        ]

        # Text line for displaying input/output
        self.textline = QLineEdit(self)
        grid.addWidget(self.textline, 0, 0, 1, 5)

        # Using a loop to generate button positions
        positions = [(i, j) for i in range(1, 5) for j in range(4)]
        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_())
```

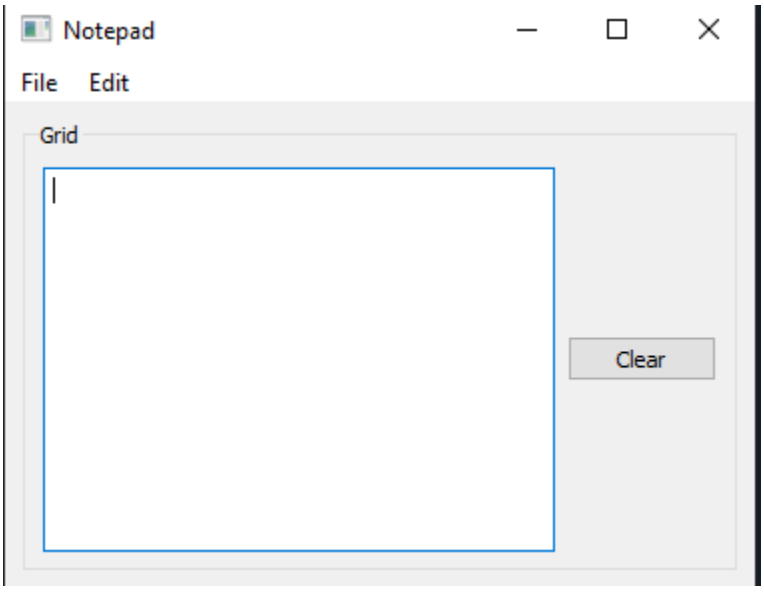
Stretched



Vbox and Hbox layout managers (Simple Notepad)

```python
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+N")
        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)
```

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

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

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

    def openFileNameDialog(self):
        options = QFileDialog.Options()
        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()


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()

    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)

        windowLayout = QVBoxLayout()
        windowLayout.addWidget(self.horizontalGroupBox)
        self.setLayout(windowLayout)
```



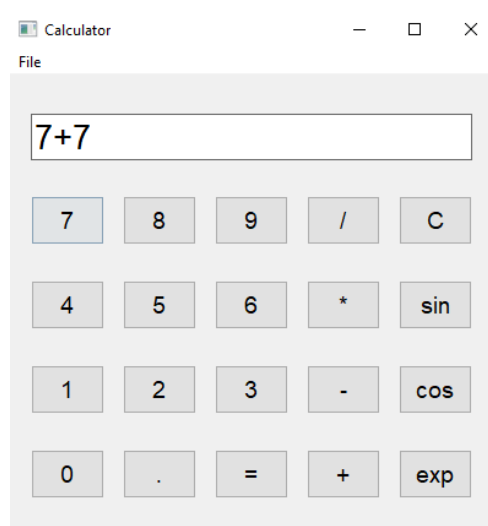
Supplementary Acitvity

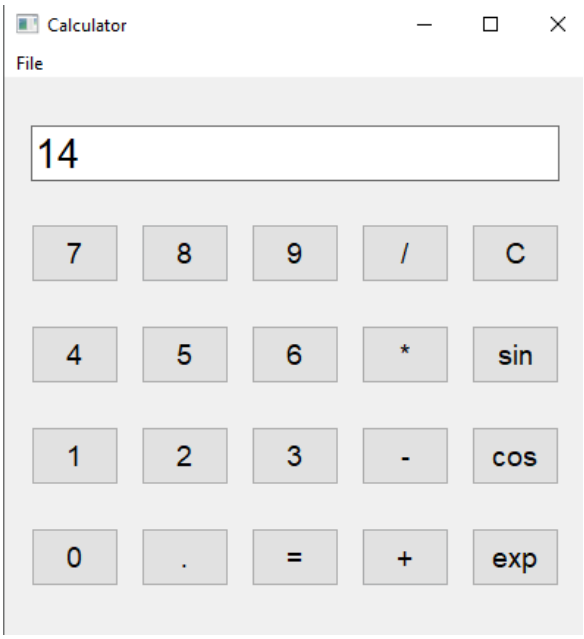
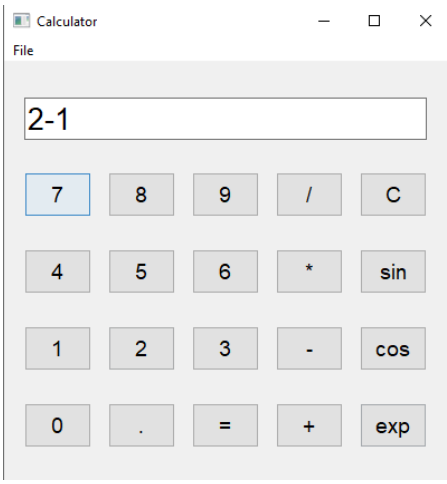
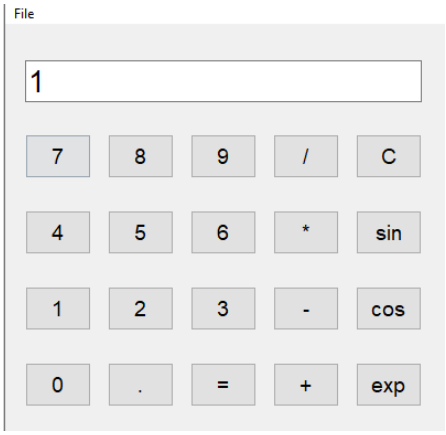
CODE:

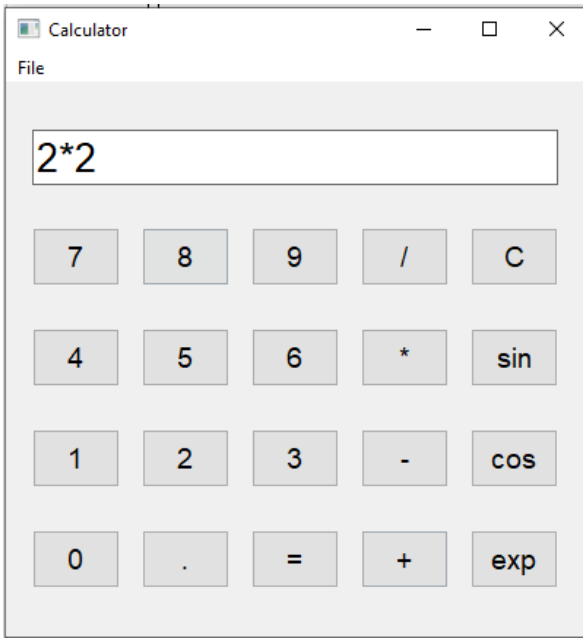
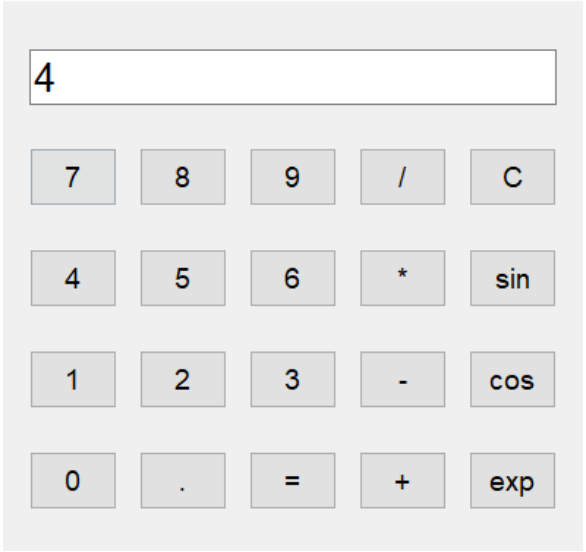
```
import sys
import math
from PyQt5.QtWidgets import (
    QMainWindow, QApplication, QWidget, QGridLayout, QLineEdit,
    QPushButton, QAction, QMessageBox
)
from PyQt5.QtGui import QFont, QIcon
from PyQt5.QtCore import QSize

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)
```

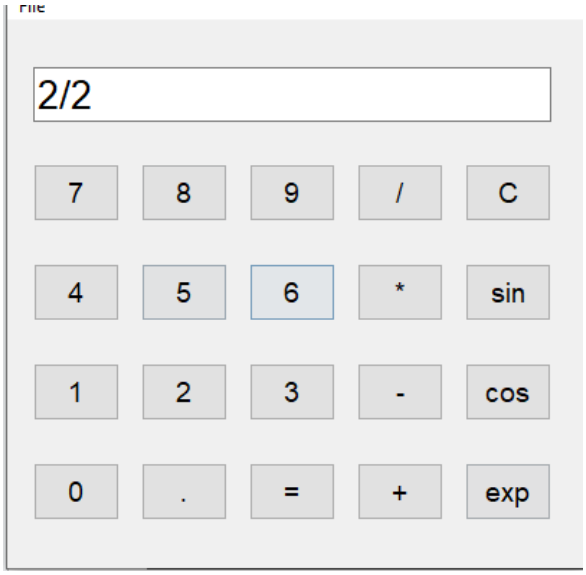
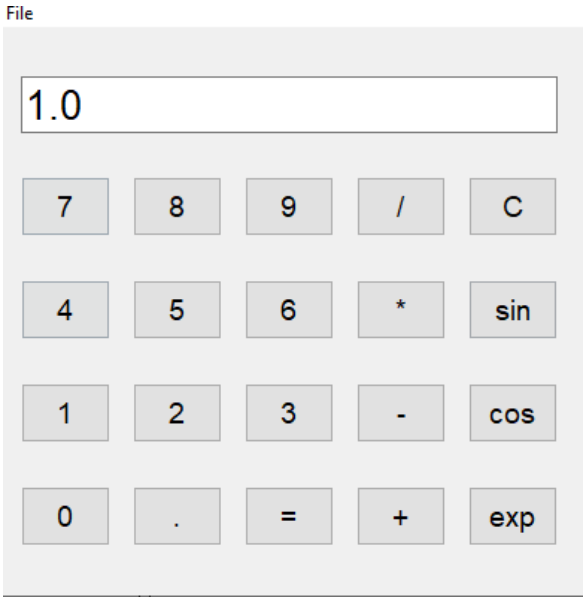
```python
        self.load_menu()

    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':
```

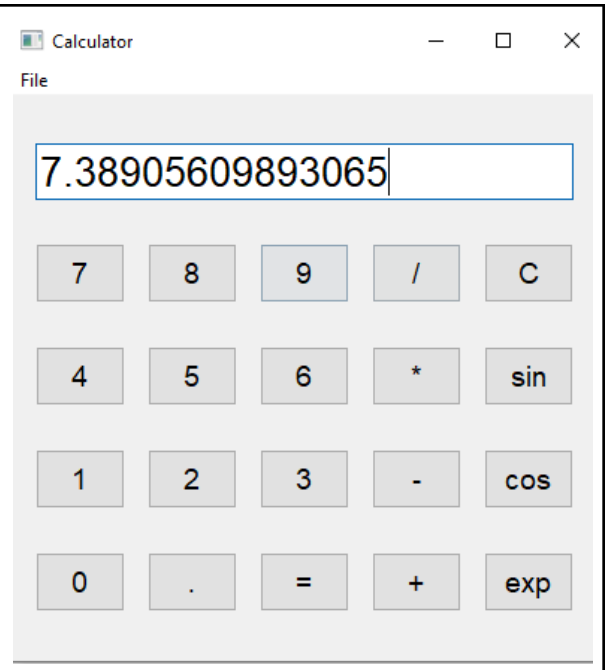
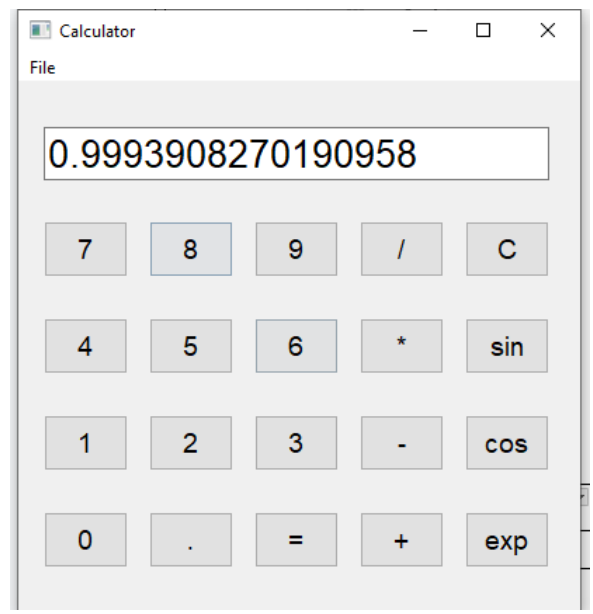
```python
    def perform_trig_or_exp(self, operation):
        try:
            value = float(self.textLine.text())
            if operation == 'sin':
                result = math.sin(math.radians(value))
            elif operation == 'cos':
                result = math.cos(math.radians(value))
            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_())
```

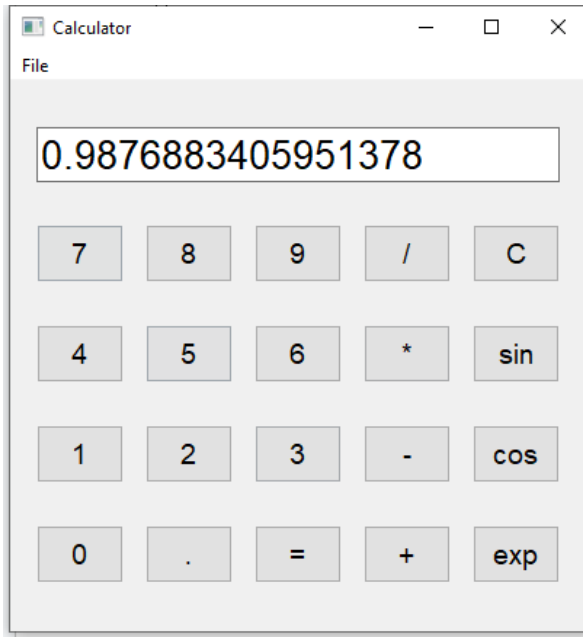
Operation	Screenshot
Addition	

	
Subtraction	 

Multiplication	 

Division	 File 2/2 7 8 9 / C 4 5 6 * sin 1 2 3 - cos 0 . = + exp  File 1.0 7 8 9 / C 4 5 6 * sin 1 2 3 - cos 0 . = + exp

Exponential (Euler Number)	
Cosine function	

Sine function	
File write	7+7 = 14 7+7 = 14 2-1 = 1 2*2 = 4 2/2 = 1.0 exp(2.0) = 7.38905609893065 cos(2.0) = 0.9993908270190958 cos(9.0) = 0.9876883405951378

Conclusion:

In this lab exercise, I learned how to create and design graphical user interfaces (GUIs) using PyQt5. I learned to work with components like buttons, text fields, and labels while using layout managers to organize them effectively. I learned using the different types of layouts like GridLayout, VBox, and HBox, allowing me to keep my UI clean and organized. I was also able to successfully create interactive interfaces with real-time feedback by fusing keyboard shortcuts and button click events. I also included file writing functionality for better practicality in using the program.