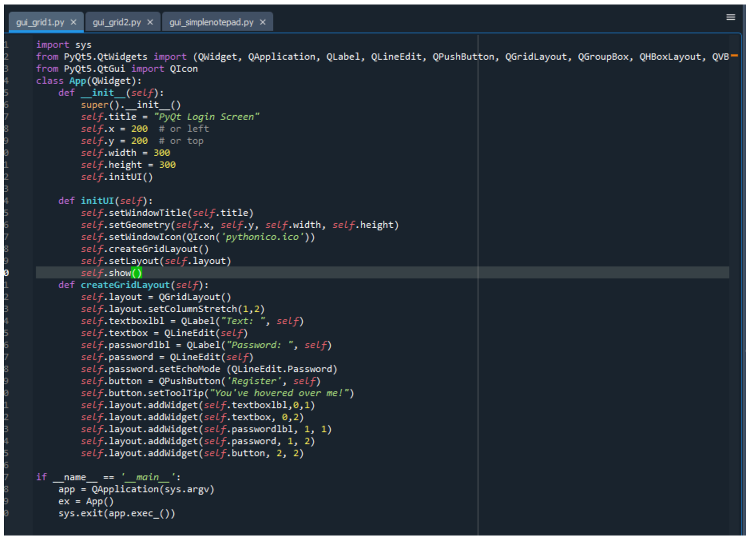
Laboratory Activity 6 - GUI Design: Layout and Styling

Reyes, Alexzander J.	28/10/2024
CPE009B/CPE21S4	Mrs. Maria Rizette Sayo

5. Procedure:

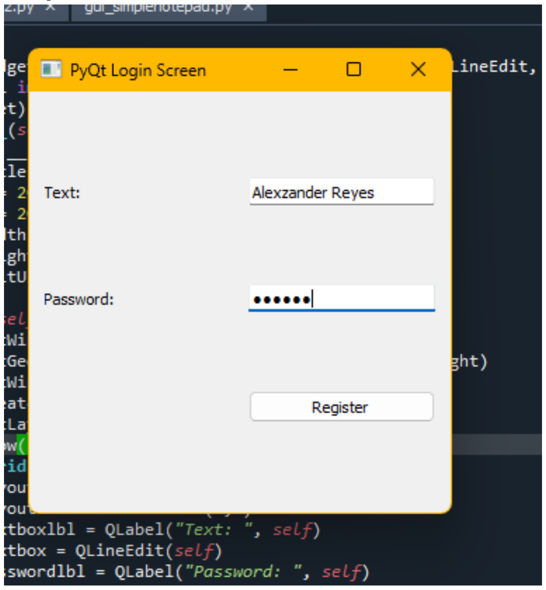
Basic Grid Layout
gui_grid1.py



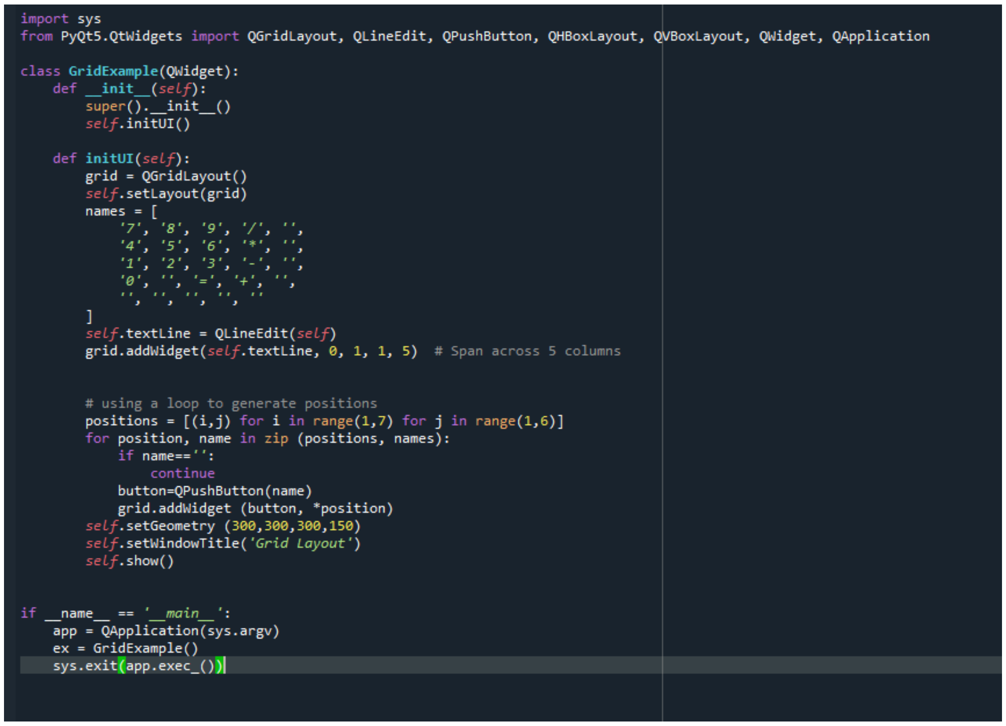
gui_grid1.py × gui_grid2.py × gui_simplenotepad.py ×

```python
import sys
from PyQt5.QtWidgets import (QWidget, QApplication, QLabel, QLineEdit, QPushButton, QGridLayout, QGroupBox, QHBoxLayout, QVB
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_())
```

Output:

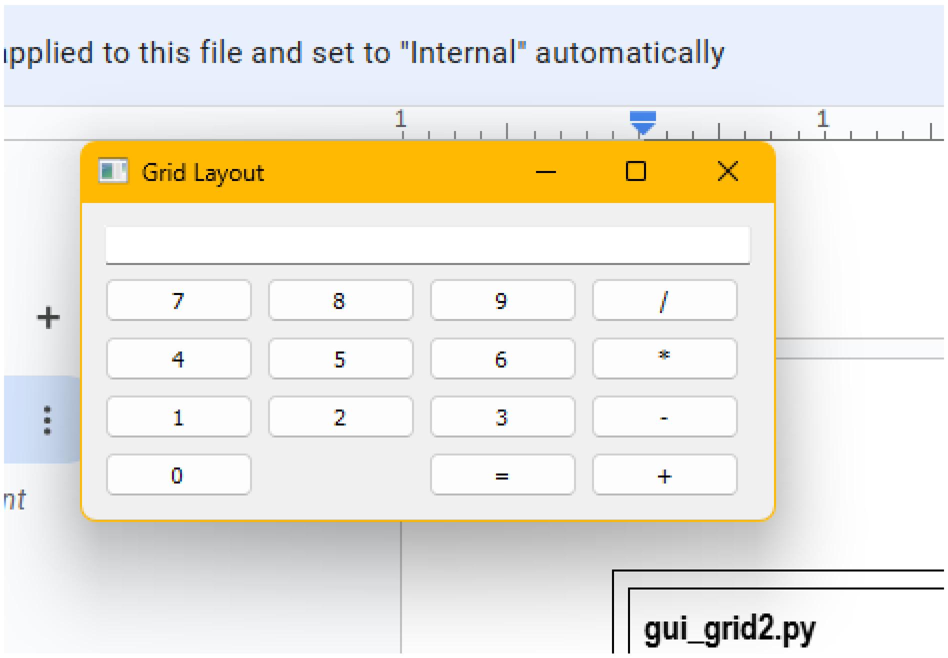


Grid Layout using Loops
gui_grid2.py



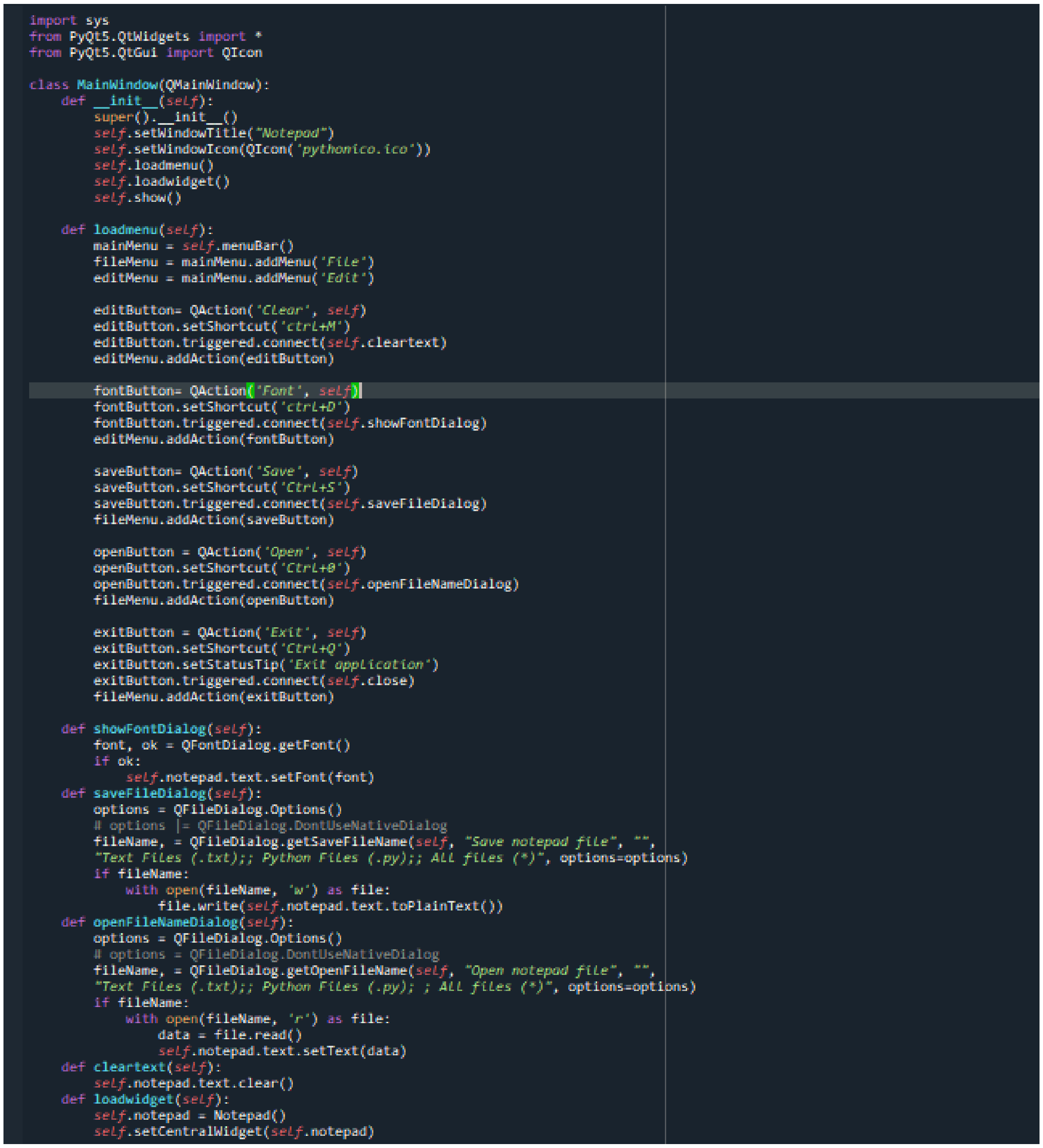
```python
import sys
from PyQt5.QtWidgets import QGridLayout, QLineEdit, QPushButton, QHBoxLayout, QVBoxLayout, QWidget, QApplication

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

    def initUI(self):
        grid = QGridLayout()
        self.setLayout(grid)
        names = [
            '7', '8', '9', '/', '',
            '4', '5', '6', '*', '',
            '1', '2', '3', '-', '',
            '0', '', '=', '+', '',
            '', '', '', '', ''
        ]
        self.textLine = QLineEdit(self)
        grid.addWidget(self.textLine, 0, 1, 1, 5)  # Span across 5 columns


        # 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_())
```

Output:



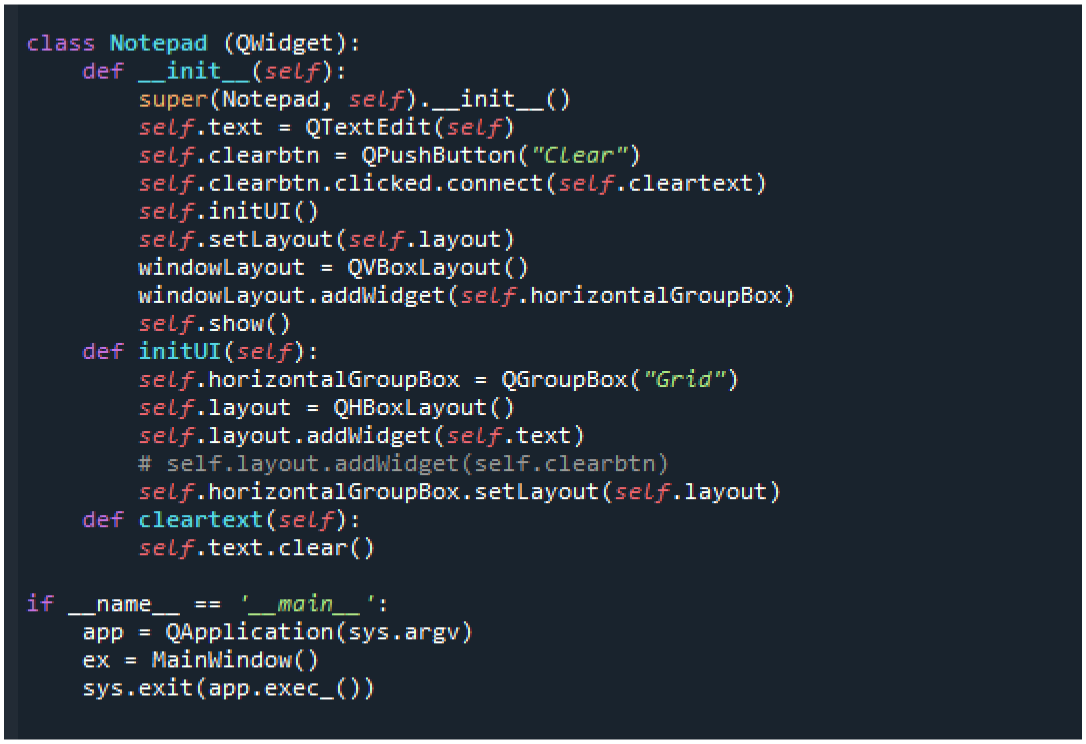
Vbox and Hbox layout managers (Simple Notepad)
gui_simplenotepad.py

MainWindow Class



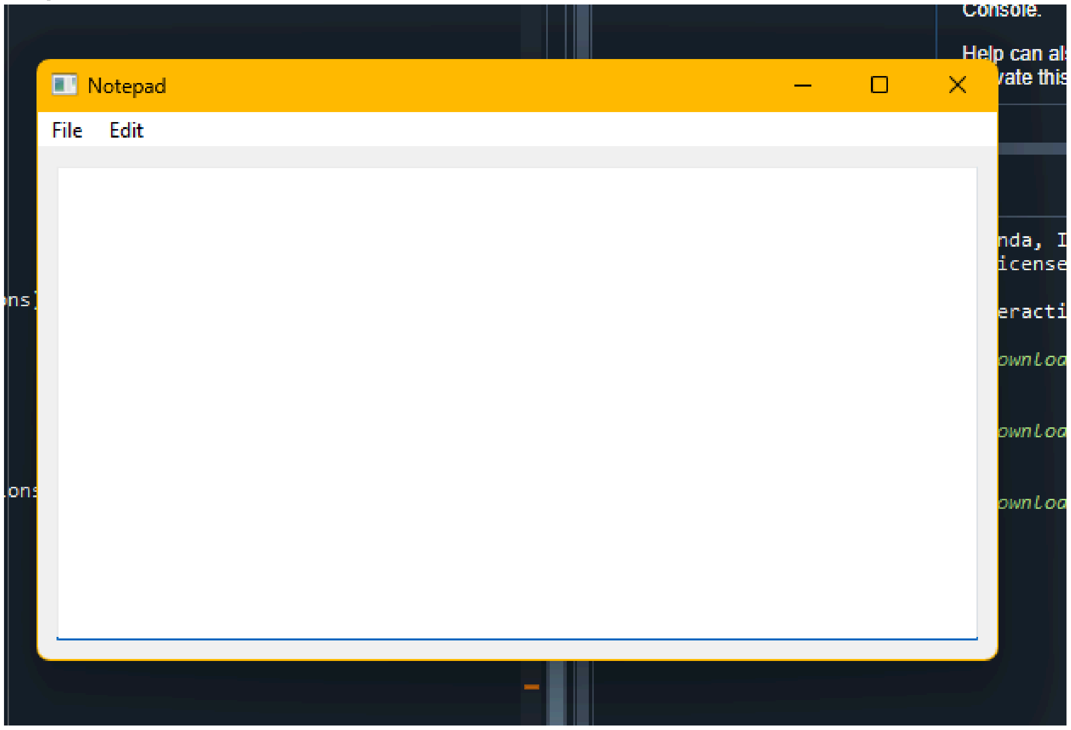
```python
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import QIcon

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+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+0')
        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)
```

Notepad Class (same file)



```
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):
        self.text.clear()

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

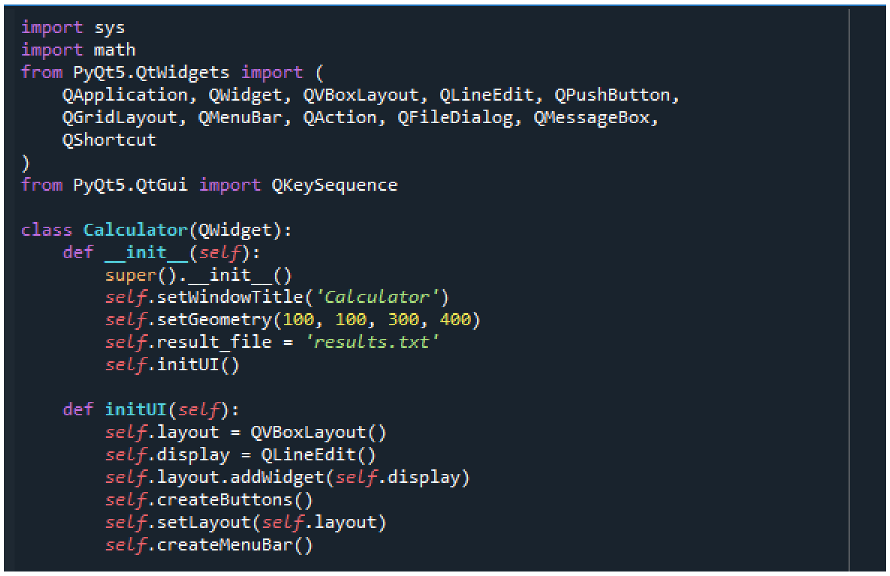
Output:



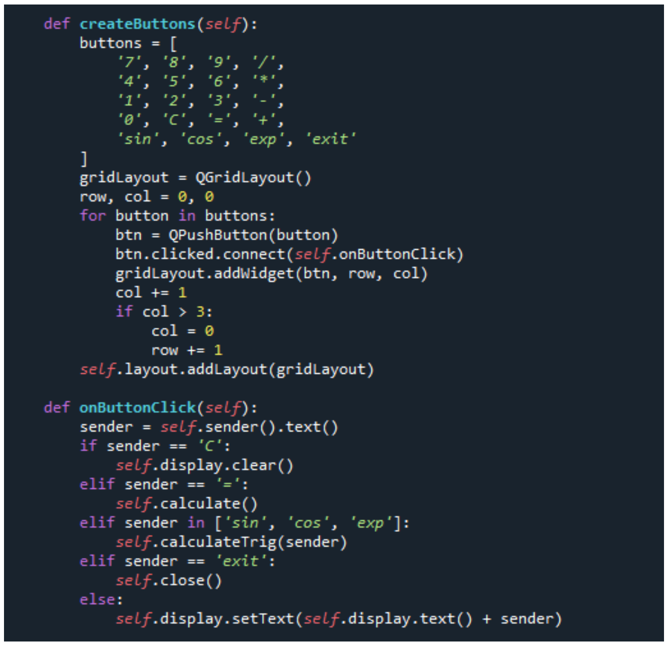
6. Supplementary Activity:

Task

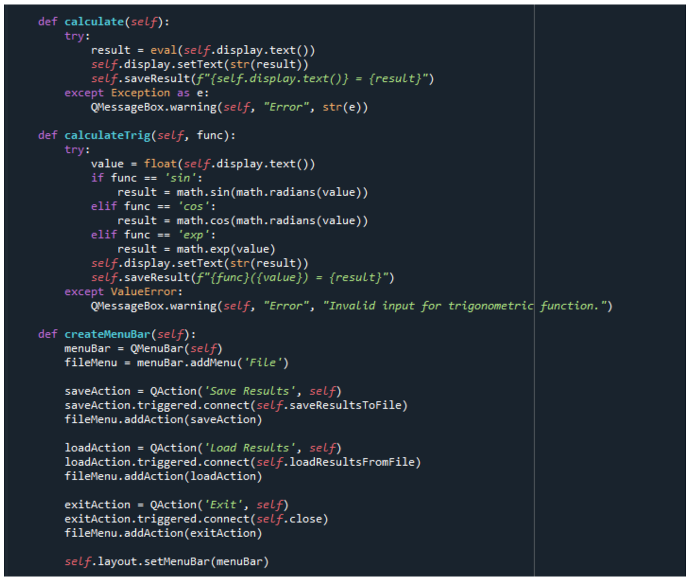
Make a calculator program that can compute perform the Arithmetic operations as well as exponential operation, sin, cosine math functions as well clearing using the C button and/or clear from a menu bar. The calculator must be able to store and retrieve the operations and results in a text file. A file menu should be available and have the option Exit which should also be triggered when ctrl+Q is pressed on the keyboard. You may refer to your calculator program in the Desktop.



```
import sys
import math
from PyQt5.QtWidgets import (
    QApplication, QWidget, QVBoxLayout, QLineEdit, QPushButton,
    QGridLayout, QMenuBar, QAction, QFileDialog, QMessageBox,
    QShortcut
)
from PyQt5.QtGui import QKeySequence

class Calculator(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Calculator')
        self.setGeometry(100, 100, 300, 400)
        self.result_file = 'results.txt'
        self.initUI()

    def initUI(self):
        self.layout = QVBoxLayout()
        self.display = QLineEdit()
        self.layout.addWidget(self.display)
        self.createButtons()
        self.setLayout(self.layout)
        self.createMenuBar()
```

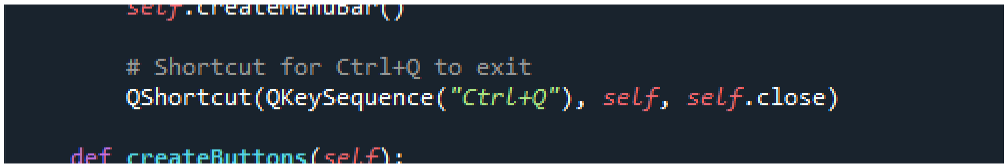


```
    def createButtons(self):
        buttons = [
            '7', '8', '9', '/',
            '4', '5', '6', '*',
            '1', '2', '3', '-',
            '0', 'C', '=', '+',
            'sin', 'cos', 'exp', 'exit'
        ]
        gridLayout = QGridLayout()
        row, col = 0, 0
        for button in buttons:
            btn = QPushButton(button)
            btn.clicked.connect(self.onButtonClick)
            gridLayout.addWidget(btn, row, col)
            col += 1
            if col > 3:
                col = 0
                row += 1
        self.layout.addLayout(gridLayout)

    def onButtonClick(self):
        sender = self.sender().text()
        if sender == 'C':
            self.display.clear()
        elif sender == '=':
            self.calculate()
        elif sender in ['sin', 'cos', 'exp']:
            self.calculateTrig(sender)
        elif sender == 'exit':
            self.close()
        else:
            self.display.setText(self.display.text() + sender)
```



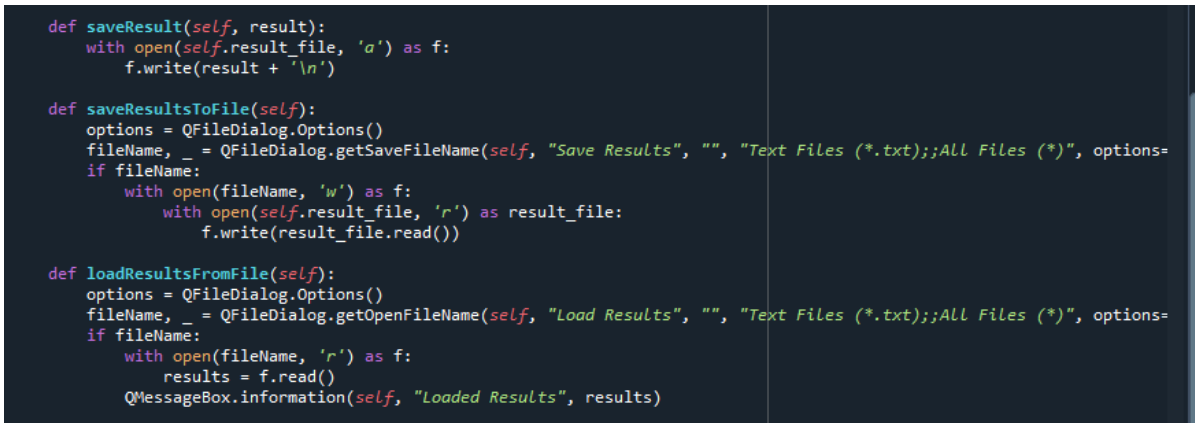
```python
    def calculate(self):
        try:
            result = eval(self.display.text())
            self.display.setText(str(result))
            self.saveResult(f"{self.display.text()} = {result}")
        except Exception as e:
            QMessageBox.warning(self, "Error", str(e))

    def calculateTrig(self, func):
        try:
            value = float(self.display.text())
            if func == 'sin':
                result = math.sin(math.radians(value))
            elif func == 'cos':
                result = math.cos(math.radians(value))
            elif func == 'exp':
                result = math.exp(value)
            self.display.setText(str(result))
            self.saveResult(f"{func}({value}) = {result}")
        except ValueError:
            QMessageBox.warning(self, "Error", "Invalid input for trigonometric function.")

    def createMenuBar(self):
        menuBar = QMenuBar(self)
        fileMenu = menuBar.addMenu('File')

        saveAction = QAction('Save Results', self)
        saveAction.triggered.connect(self.saveResultsToFile)
        fileMenu.addAction(saveAction)

        loadAction = QAction('Load Results', self)
        loadAction.triggered.connect(self.loadResultsFromFile)
        fileMenu.addAction(loadAction)

        exitAction = QAction('Exit', self)
        exitAction.triggered.connect(self.close)
        fileMenu.addAction(exitAction)

        self.layout.setMenuBar(menuBar)
```

Exit Option (Ctrl+Q)



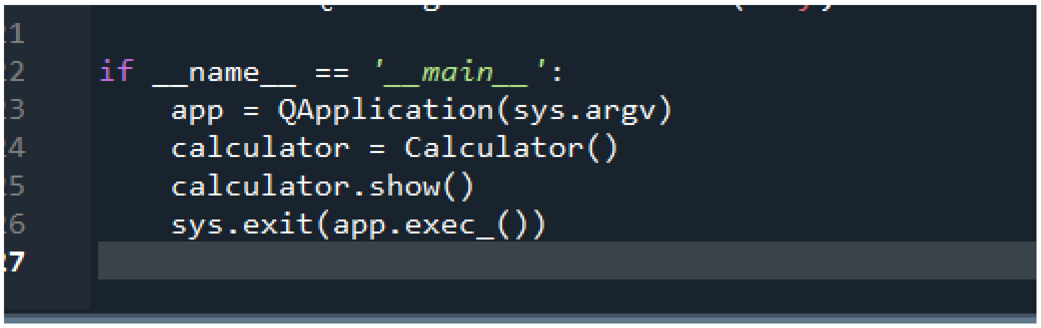
```python
        self.createMenuBar()

        # Shortcut for Ctrl+Q to exit
        QShortcut(QKeySequence("Ctrl+Q"), self, self.close)

    def createButtons(self):
```

Save Result



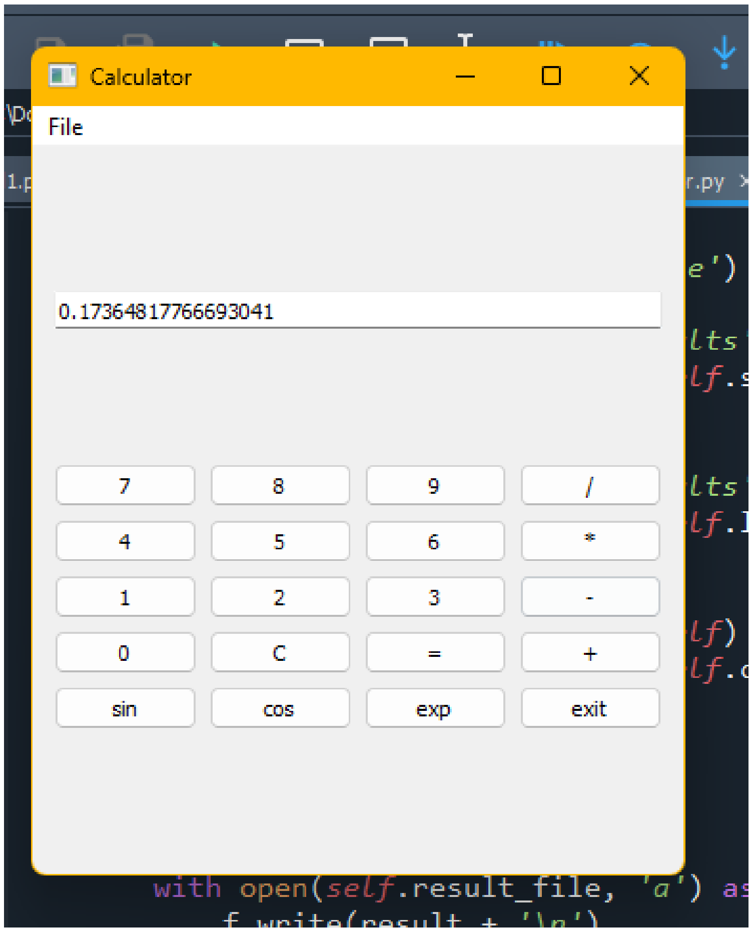
```python
    def saveResult(self, result):
        with open(self.result_file, 'a') as f:
            f.write(result + '\n')

    def saveResultsToFile(self):
        options = QFileDialog.Options()
        fileName, _ = QFileDialog.getSaveFileName(self, "Save Results", "", "Text Files (*.txt);;All Files (*)", options=
        if fileName:
            with open(fileName, 'w') as f:
                with open(self.result_file, 'r') as result_file:
                    f.write(result_file.read())

    def loadResultsFromFile(self):
        options = QFileDialog.Options()
        fileName, _ = QFileDialog.getOpenFileName(self, "Load Results", "", "Text Files (*.txt);;All Files (*)", options=
        if fileName:
            with open(fileName, 'r') as f:
                results = f.read()
            QMessageBox.information(self, "Loaded Results", results)
```

Code to run the GUI

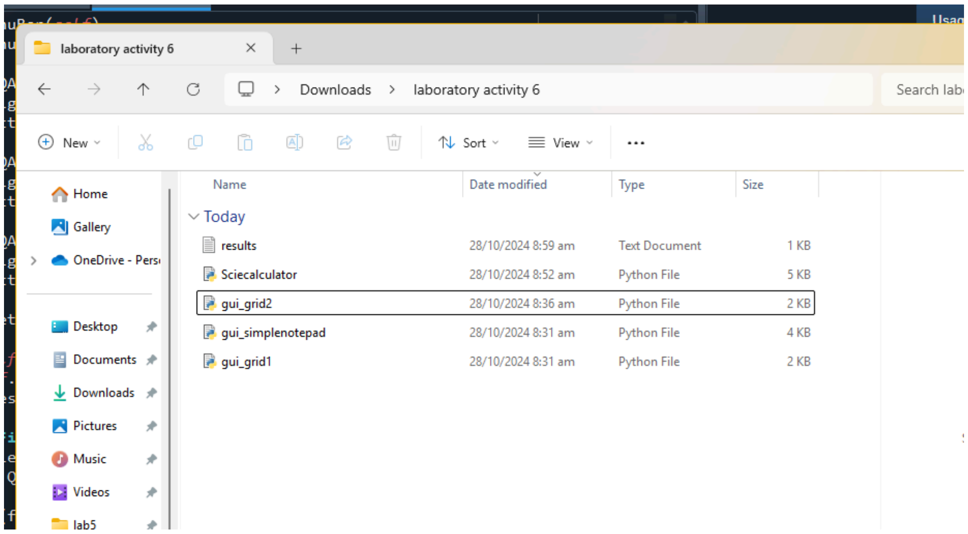


```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    calculator = Calculator()
    calculator.show()
    sys.exit(app.exec_())
```

Output:

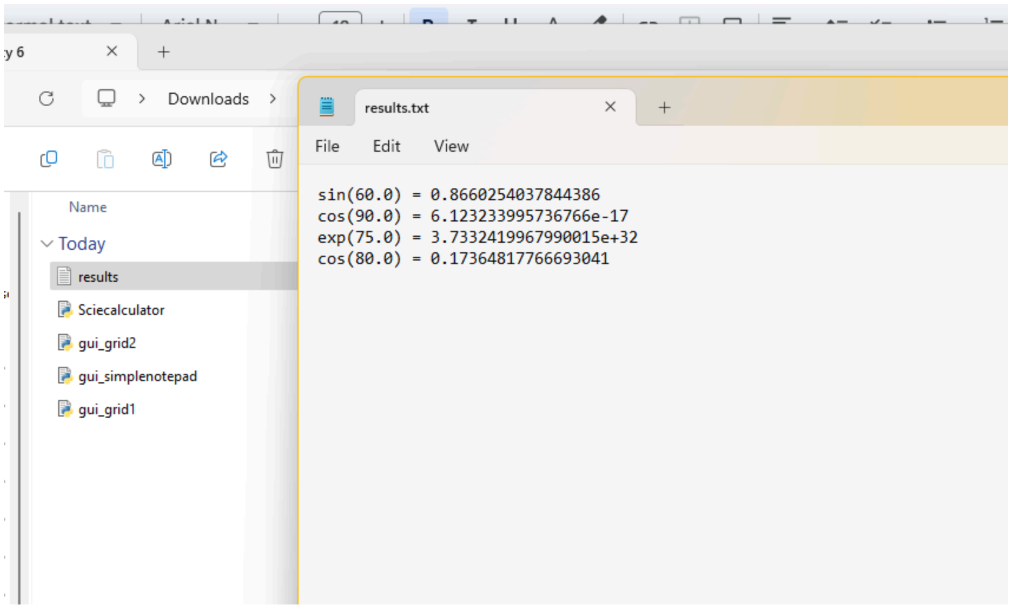


File Folder



Name	Date modified	Type	Size
Today			
results	28/10/2024 8:59 am	Text Document	1 KB
Sciecalculator	28/10/2024 8:52 am	Python File	5 KB
gui_grid2	28/10/2024 8:36 am	Python File	2 KB
gui_simplenotepad	28/10/2024 8:31 am	Python File	4 KB
gui_grid1	28/10/2024 8:31 am	Python File	2 KB

Result Text File



```
sin(60.0) = 0.8660254037844386
cos(90.0) = 6.123233995736766e-17
exp(75.0) = 3.7332419967990015e+32
cos(80.0) = 0.17364817766693041
```

7. Conclusion:

In this activity, we focus on our calculator application's graphical user interface (GUI) design, highlighting layout and styling to improve user experience. Utilizing a vertical box layout, we successfully arranged the buttons and display in a clear and user-friendly manner. This layout improves navigation, which makes it easy for users to enter calculations without being confused. The button grid was thoughtfully created to enable fast access by gathering related tasks together. Furthermore, in order to increase exposure and engagement, we used a visually appealing style with unique button colors and sizes. The menu bar offers users choices to save and load results with ease, giving file management a more organized approach.This design is enhanced by the underlying computer programming, which divides functionality into understandable, controllable parts. Signal-slot connections link each button on the calculator to a distinct function, guaranteeing that user inputs result in the right answers. For computations, we used trigonometric functions to perform certain mathematical operations and the eval function to dynamically compute expressions entered in the display. In order to properly handle erroneous inputs and give consumers insightful feedback, error handling systems are in place. Usability is further improved by file management tools that enable results to be saved and loaded. All things considered, the calculator's functionality and user-friendliness are guaranteed by the combination of careful GUI design and well-organized code, which improves user engagement and satisfaction.

8. Assessment Rubric: