Laboratory Activity No. 6	
GUI Design: Layout and Styling	
Course Code: CPE009	**Program:** Computer Engineering
Course Title: Object-Oriented Programming	**Date Performed:** 10/28/24
Section: CPE21S4	**Date Submitted:** 10/28/24
Name(s): Anna Marie Zolina	**Instructor:** Prof. Maria Rizette Sayo

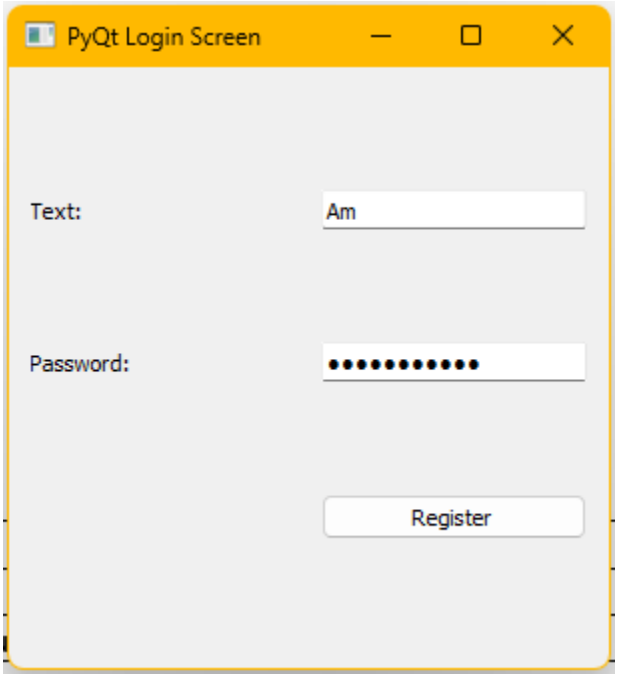
6. Output

Basic Grid Layout
Source Code:

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

class App(QWidget):

   def __init__(self):
       super().__init__()
       self.title= "PyQt Login Screen"
       self.x = 200
       self.y = 200
       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:



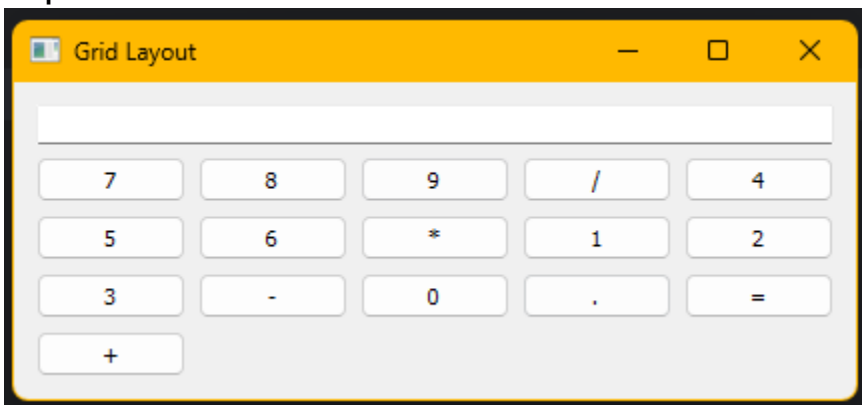
Grid Layout using Loops

Source Code:

```
#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)
```

```
        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)

MainWindow Class

```
import sys
from PyQt5.QWidgets import*
from PyQt5.QtGui impport QIcon

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

        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')
```

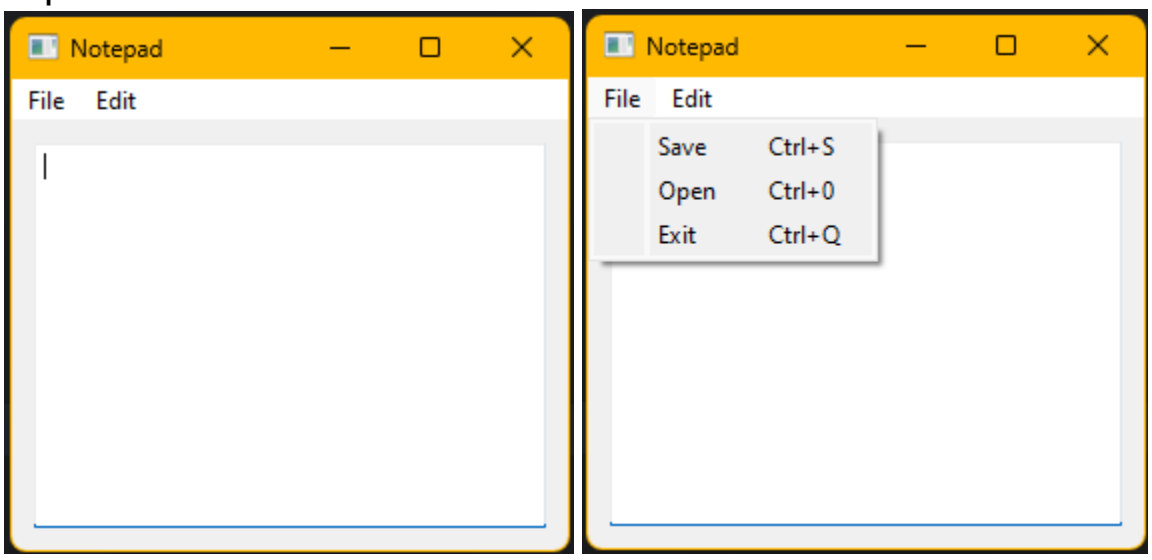
```
        editButton.triggered.connect(self.cleartext)
        editMenu.addAction(fontButton)

        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()
        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()
        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):
```

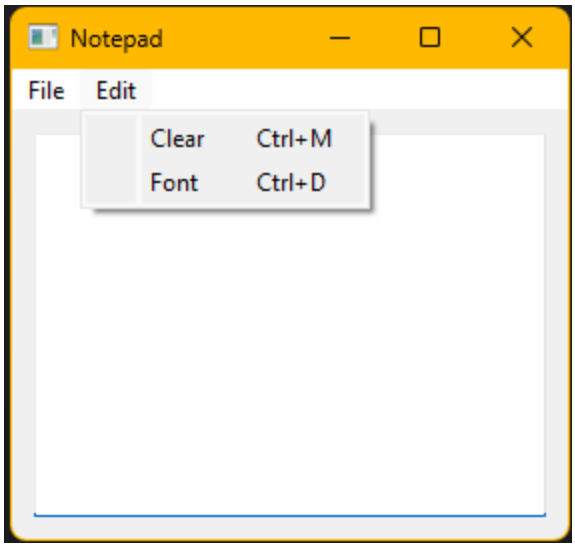
```
        self.notepad = Notepad()
        self.setCentralWidget(self.notepad)
```

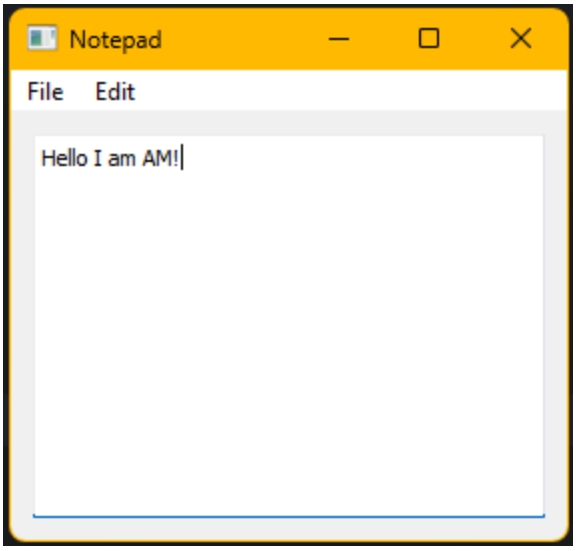
Notepad Class
Source Code:

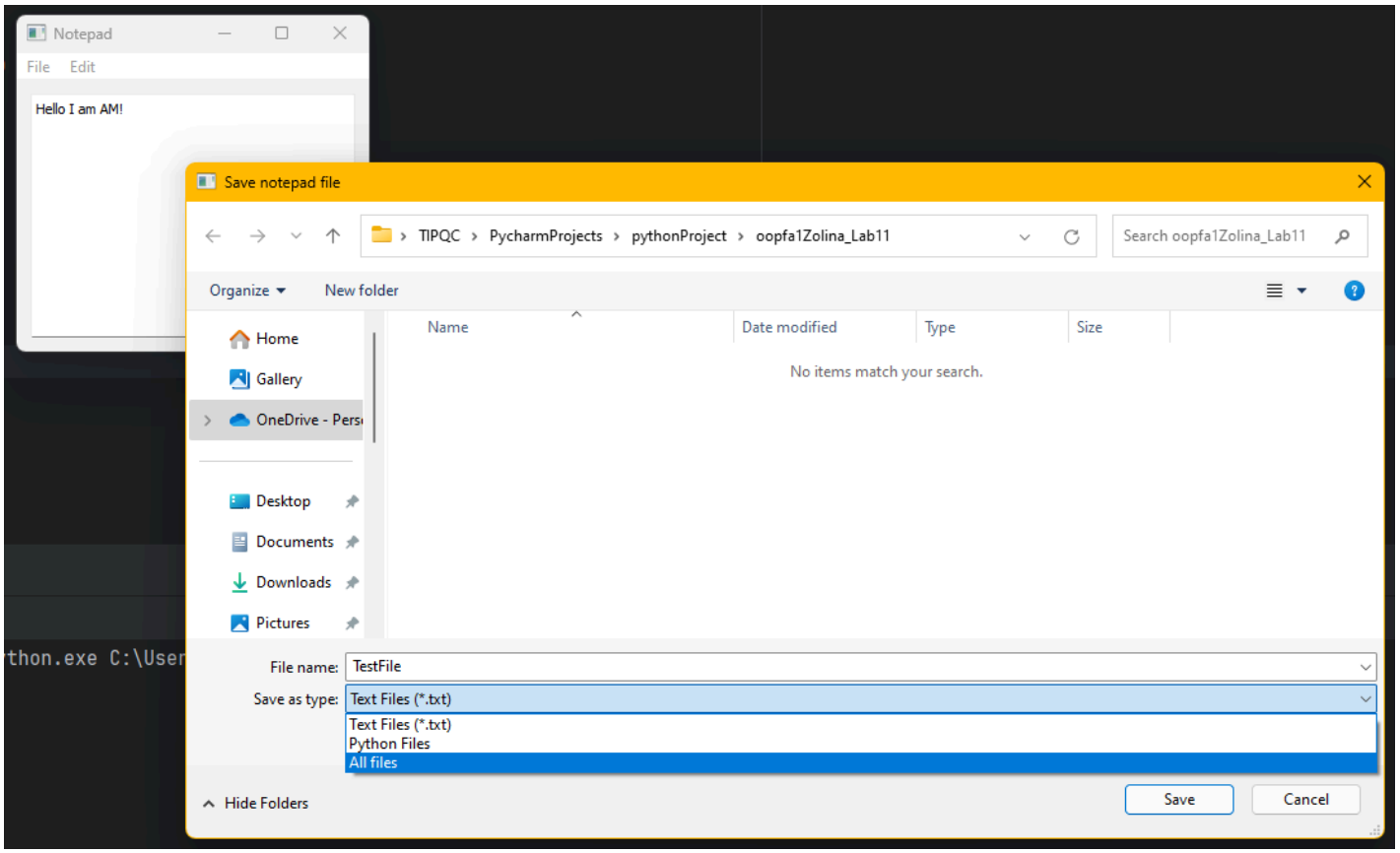
```
class Notepad(QWidget):

    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.horizontalGroupBox.setLayout(self.layout)

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

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

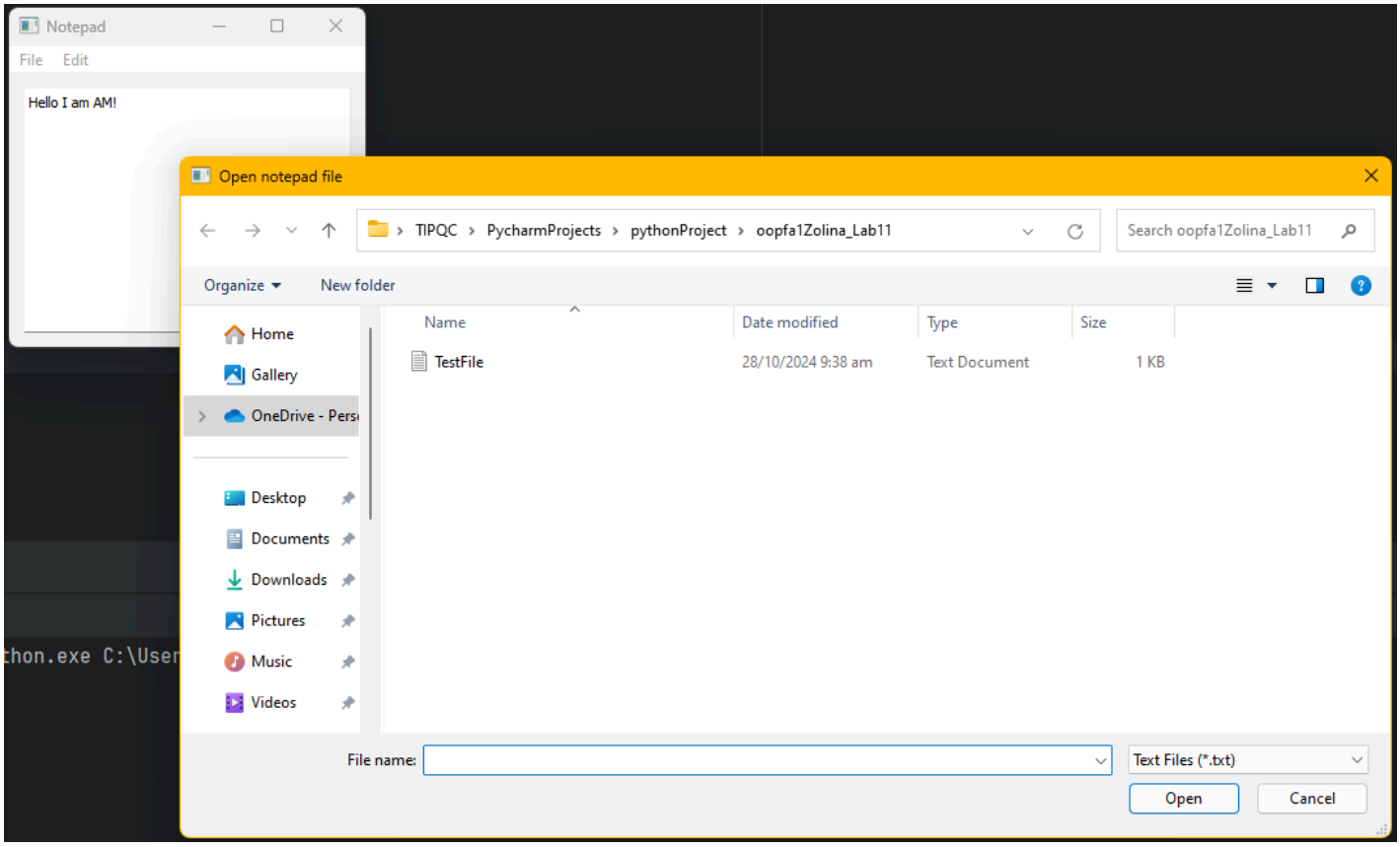
Output:

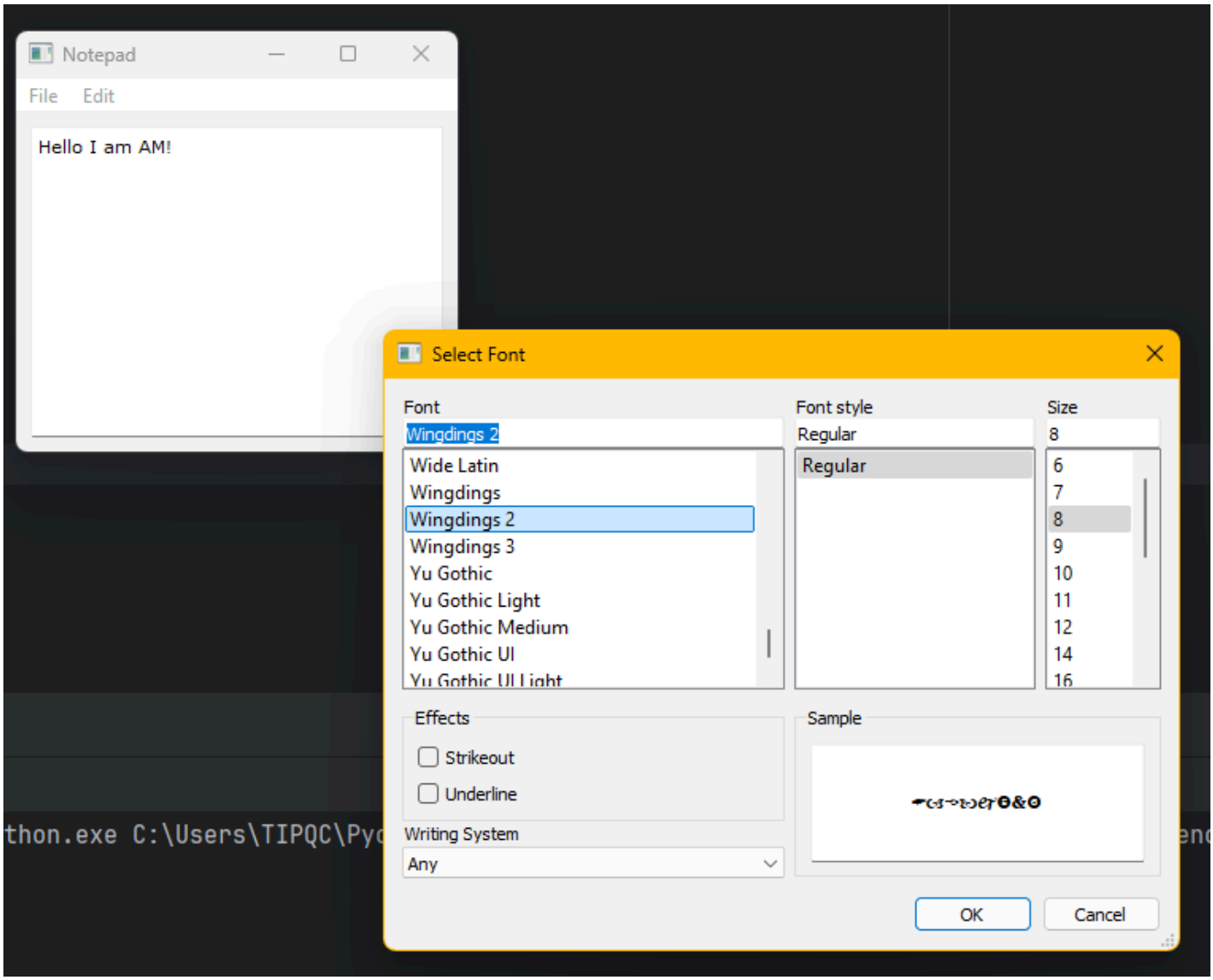


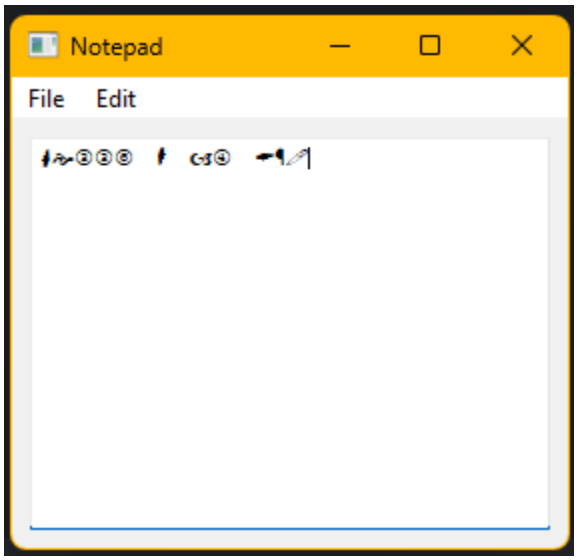

Notepad
File
Edit
Clear Ctrl+M
Font Ctrl+D

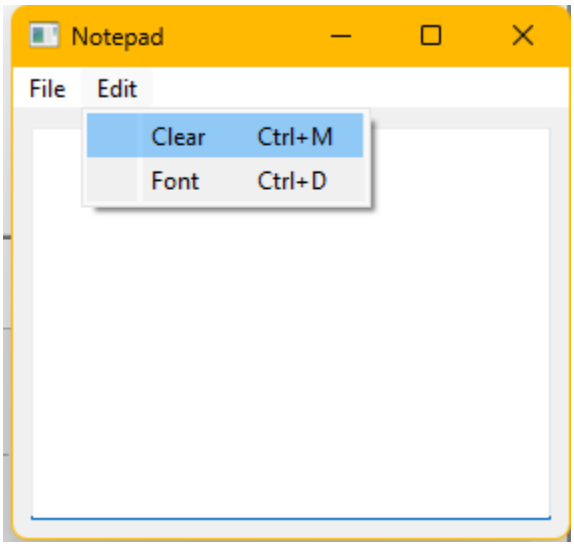

Notepad
File
Edit
Hello I am AM!


Notepad
File
Edit
Hello I am AM!
Save notepad file
TIPQC › PycharmProjects › pythonProject › oopfa1Zolina_Lab11
Search oopfa1Zolina_Lab11
Organize
New folder
Home
Gallery
OneDrive - Pers
Desktop
Documents
Downloads
Pictures
Name
Date modified
Type
Size
No items match your search.
File name: TestFile
Save as type: Text Files (*.txt)
Text Files (*.txt)
Python Files
All files
Hide Folders
Save
Cancel
thon.exe C:\User


Notepad
File Edit
Hello I am AM!
Open notepad file
TIPQC › PycharmProjects › pythonProject › oopfa1Zolina_Lab11
Search oopfa1Zolina_Lab11
Organize
New folder
Home
Gallery
OneDrive - Pers
Desktop
Documents
Downloads
Pictures
Music
Videos
Name
Date modified
Type
Size
TestFile
28/10/2024 9:38 am
Text Document
1 KB
File name:
Text Files (*.txt)
Open
Cancel
thon.exe C:\User







7. Supplementary Activity:

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.

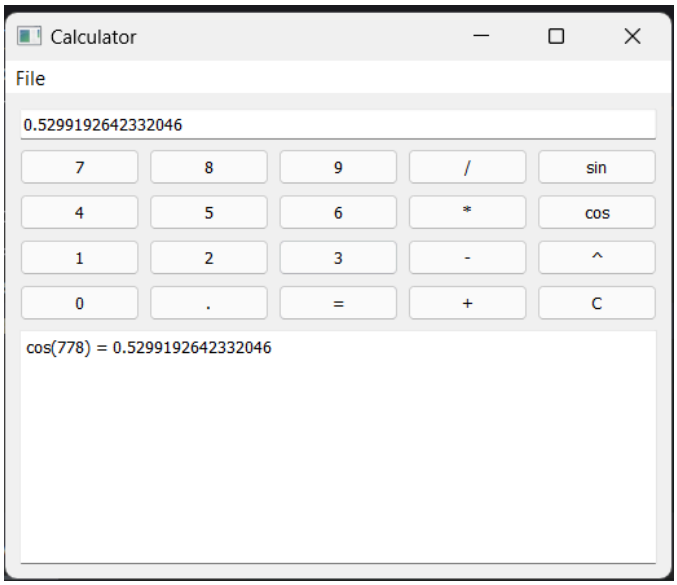
SOURCE CODE:

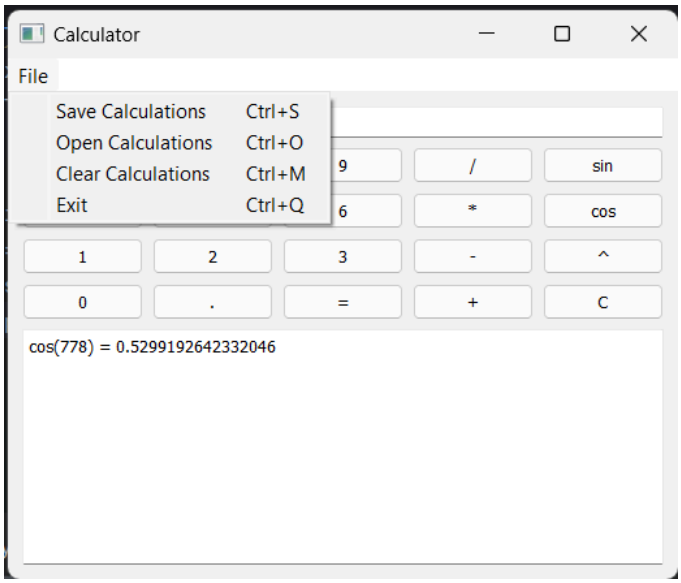
```
import sys
from PyQt5.QtWidgets import (
    QWidget, QGridLayout, QLineEdit, QPushButton, QVBoxLayout, QApplication,
    QMainWindow, QAction, QFileDialog, QTextEdit, QMenuBar
)
from PyQt5.QtGui import QIcon
import math

class CalculatorApp(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Calculator")
        self.setGeometry(300, 300, 400, 400)
        self.initUI()

    def initUI(self):
        mainWidget = QWidget(self)
        mainLayout = QVBoxLayout()

        # Display
        self.textLine = QLineEdit(self)
        mainLayout.addWidget(self.textLine)

        # Grid Layout for buttons
        grid = QGridLayout()
        mainLayout.addLayout(grid)

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

        # Add buttons to grid
        positions = [(i, j) for i in range(5) for j in range(5)]
        for position, name in zip(positions, names):
            button = QPushButton(name)
            button.clicked.connect(self.onButtonClick)
            grid.addWidget(button, *position)

        # Text area for calculations
        self.calculations = QTextEdit()
        mainLayout.addWidget(self.calculations)

        # Set main layout
        mainWidget.setLayout(mainLayout)
        self.setCentralWidget(mainWidget)

        # Menu
        self.createMenu()
```

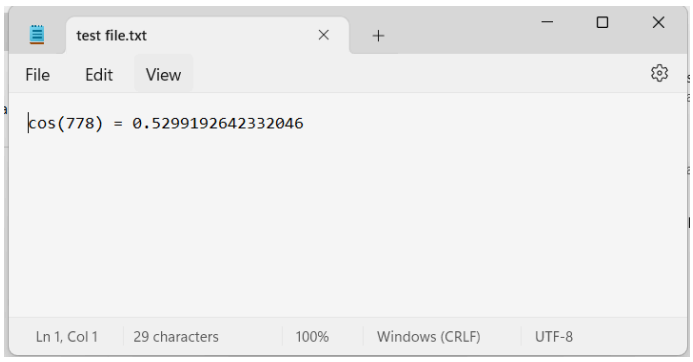
```
    def createMenu(self):
        mainMenu = QMenuBar(self)

        # File menu
        fileMenu = mainMenu.addMenu('File')

        # Save action
        saveAction = QAction('Save Calculations', self)
        saveAction.setShortcut('Ctrl+S')
        saveAction.triggered.connect(self.saveCalculations)
        fileMenu.addAction(saveAction)

        # Open action
        openAction = QAction('Open Calculations', self)
        openAction.setShortcut('Ctrl+O')
        openAction.triggered.connect(self.openCalculations)
        fileMenu.addAction(openAction)

        # Clear action
        clearAction = QAction('Clear Calculations', self)
        clearAction.setShortcut('Ctrl+M')
        clearAction.triggered.connect(self.clearCalculations)
        fileMenu.addAction(clearAction)

        # Exit action
        exitAction = QAction('Exit', self)
        exitAction.setShortcut('Ctrl+Q')
        exitAction.triggered.connect(self.close)
        fileMenu.addAction(exitAction)

        self.setMenuBar(mainMenu)

    def keyPressEvent(self, event):
        key = event.text()
        if key in '0123456789.+-*/':
            self.textLine.setText(self.textLine.text() + key)
        elif key == '=':
            self.evaluateExpression()
        elif key.lower() == 'c':
            self.clearCalculations()  # Calls the clear function
        elif key == '^':
            self.textLine.setText(self.textLine.text() + '**')

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

        if button_text == 'C':
            # Clear the input field
            self.clearCalculations()  # Calls the clear function
        elif button_text == '=':
            self.evaluateExpression()
        elif button_text == 'sin':
            self.calculateTrig('sin')
        elif button_text == 'cos':
            self.calculateTrig('cos')
        elif button_text == '^':
            # Append exponentiation symbol for power operation
            current_text = self.textLine.text()
            self.textLine.setText(current_text + '**')
        else:
            # Append the button text to the current input
```

```
            current_text = self.textLine.text()
            self.textLine.setText(current_text + button_text)

    def evaluateExpression(self):
        # Calculate and display result
        expression = self.textLine.text()
        expression = expression.replace("^", "**")
        try:
            result = str(eval(expression))  # Direct calculation
            self.textLine.setText(result)
            self.calculations.append(f"{expression} = {result}")
        except:
            self.textLine.setText("Error")  # Display error for invalid expressions

    def calculateTrig(self, func):
        # Calculate trigonometric function in degrees
        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("Error")  # Handle invalid input for trig functions

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

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

    def clearCalculations(self):
        self.textLine.clear()  # Clear the input field
        self.calculations.clear()  # Clear the calculations text area

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

OUTPUT:







8. Conclusion

I learned a lot about creating a graphical user interface (GUI) with multiple widgets and comparing it to the features of a notepad application through this lab exercise. I gained knowledge about how to properly arrange buttons and choose the right sizes for the layout of the program. I also experimented with various Q functions, like `QGridLayout`, which allows for precise element placement, and `QVBoxLayout`, which arranges the primary components in a vertical posiytion. Although I believe I did well overall in this exercise, I am aware that I still have a lot to learn before I can completely comprehend the underlying code and functions. I want to improve my ability to write more dynamic and effective GUI programs with more practice.

9. Assessment Rubric