Activity Name #6 GUI Design: Layout and Styling	
Santos, Ma. Kassandra Nicole D.	6/27/2019
Course/Section: CPE21S4	Ma'am Rizette Sayo

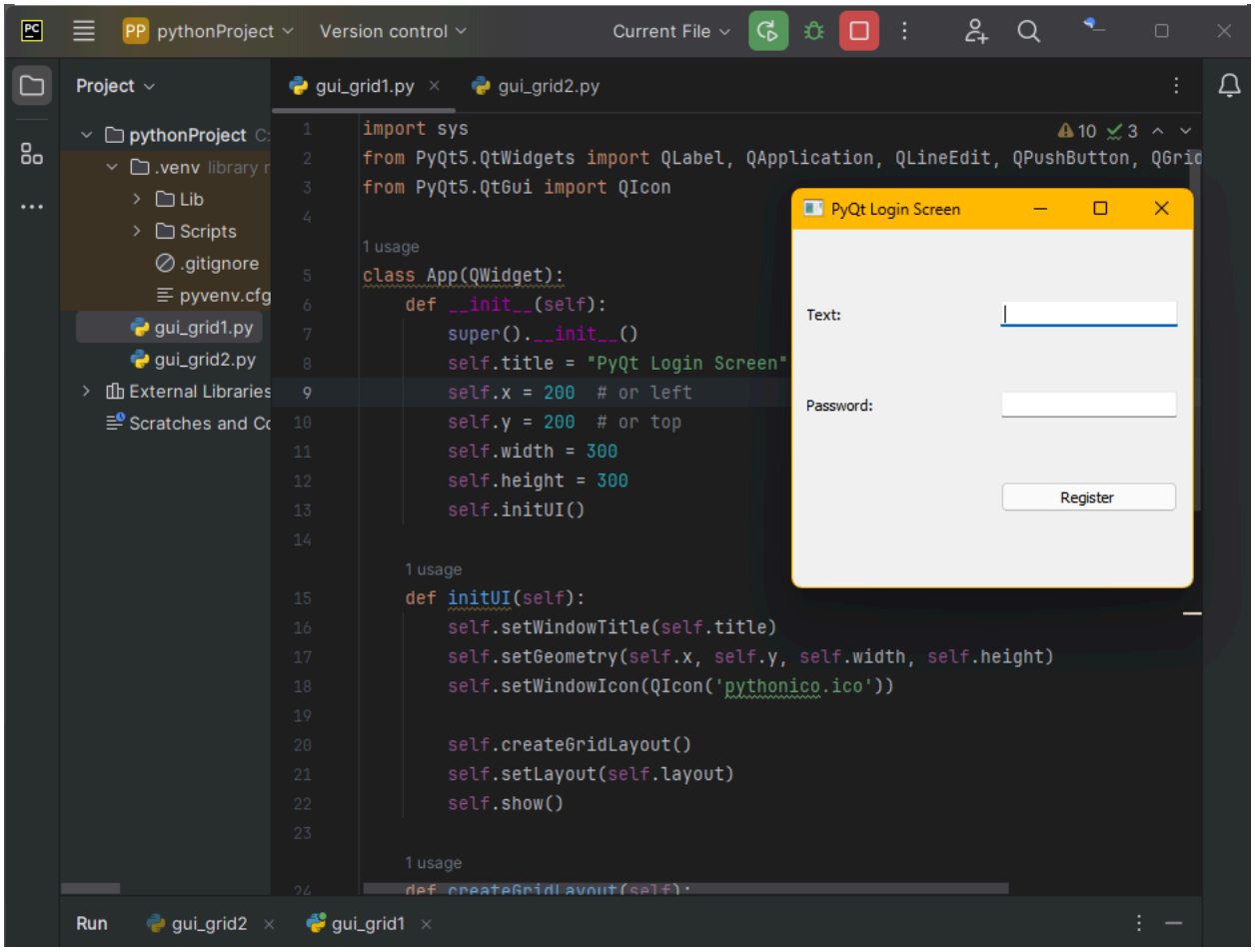
BASIC GRID LAYOUT

The Code:

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

The Output:



Observation:

After executing the code; the output shows a login screen that involves a text and a password wherein the password is hidden. You can interact with output however there is no follow up action once you click register

---

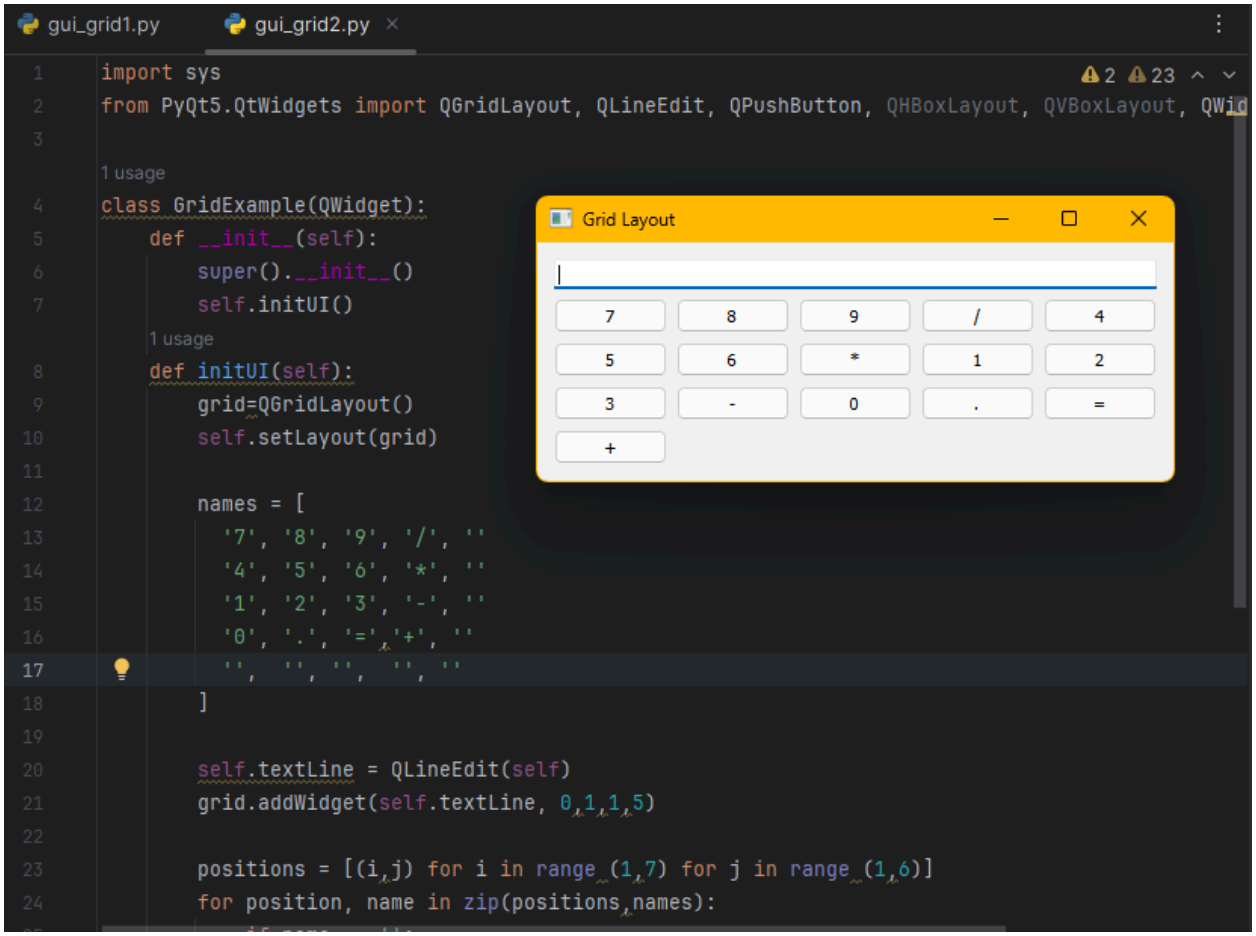
GRID LAYOUT USING LOOPS

The Code:

```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)

        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_())
```

The Output



Observation:

Upon executing the code, a calculator showed up wherein you can interact with it. However the design of the calculator is different compared to the other calculators that you usually see, the numbers do not align with each other even though I made sure that code that I followed was correct.

VBOX AND HBOX MANAGERS:

The Code:

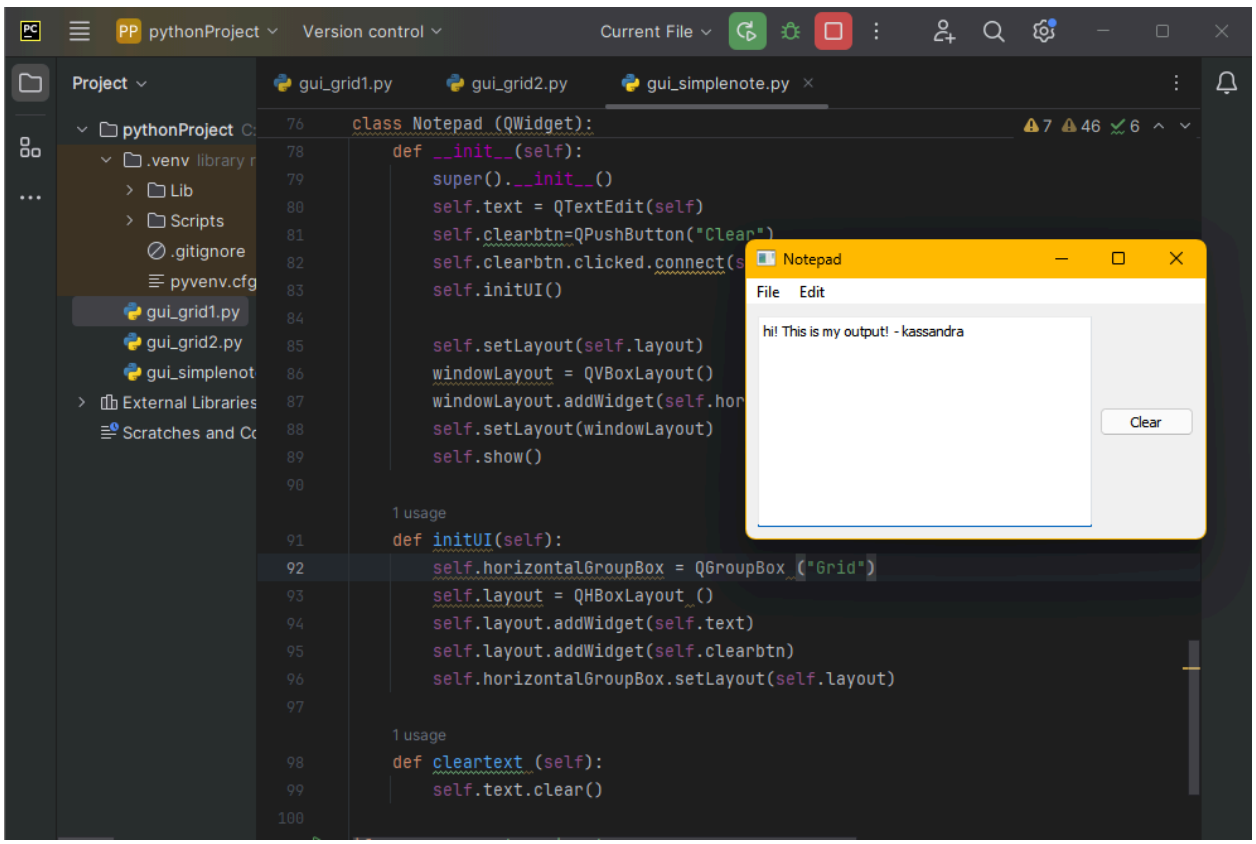
```python
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import QIcon
```

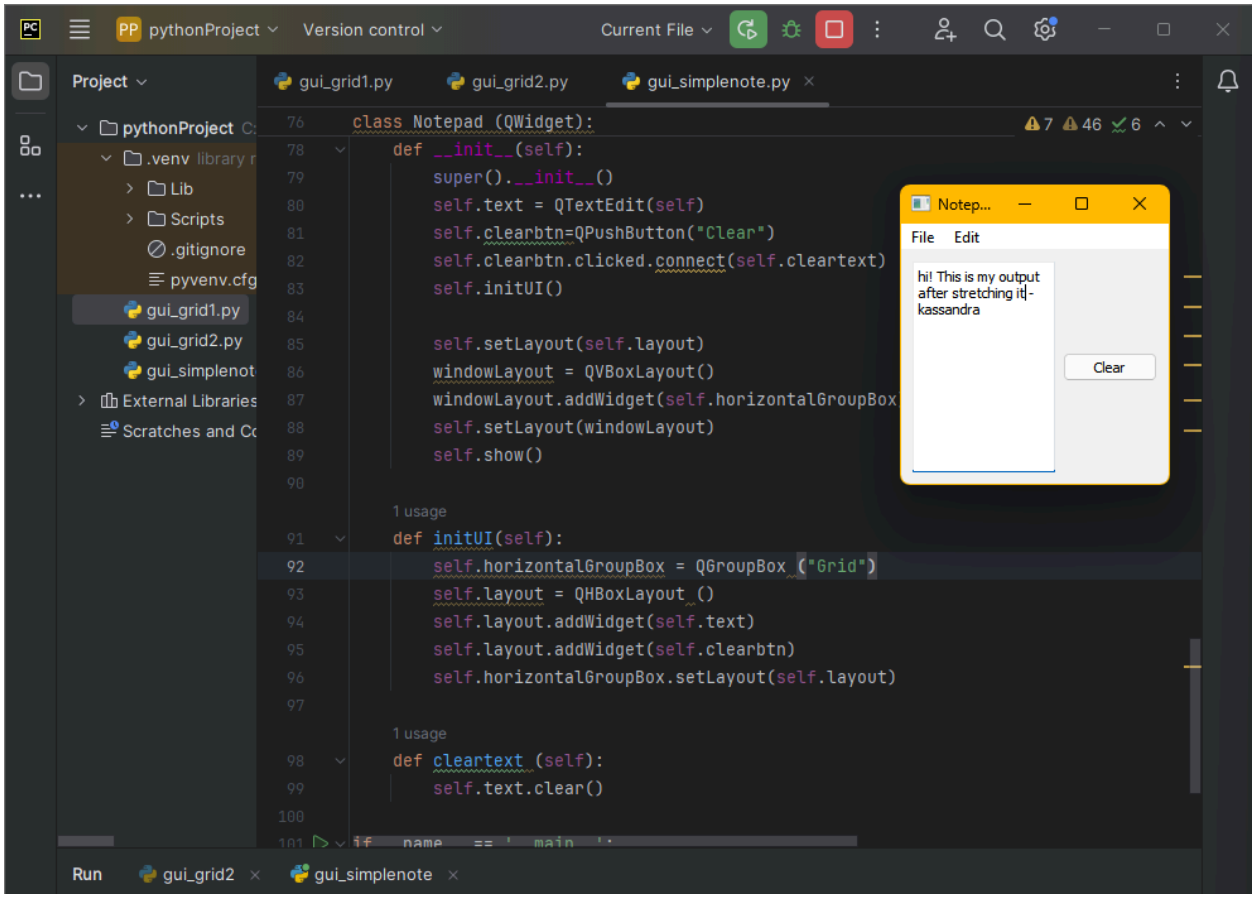
```python
class MainWindow (QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Notepad")
        self.setWindowIcon(QIcon('pythonico.ico'))
        self.loadmenu()

        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()
        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)

class Notepad (QWidget):

    def __init__(self):
        super().__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.setLayout(windowLayout)
        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_())
```

The Output:





Observation:

Upon executing the code, the program shows a regular notepad with a clear button on the right side. All of the buttons, including the clear button, work and are functional according to their purpose. When stretching the notepad, the text adjusts to the available space without disappearing or having any problems.

SUPPLEMENTARY ACTIVITY

The CODE

```python
import sys
```

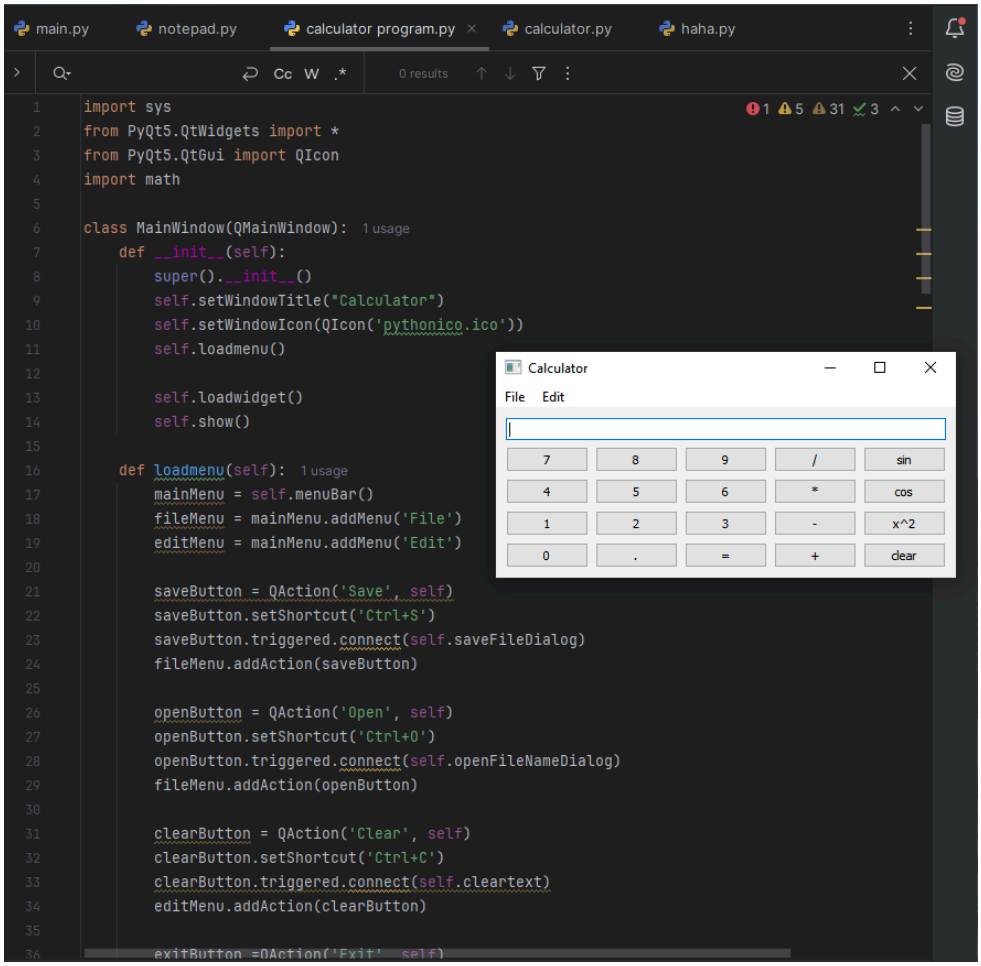
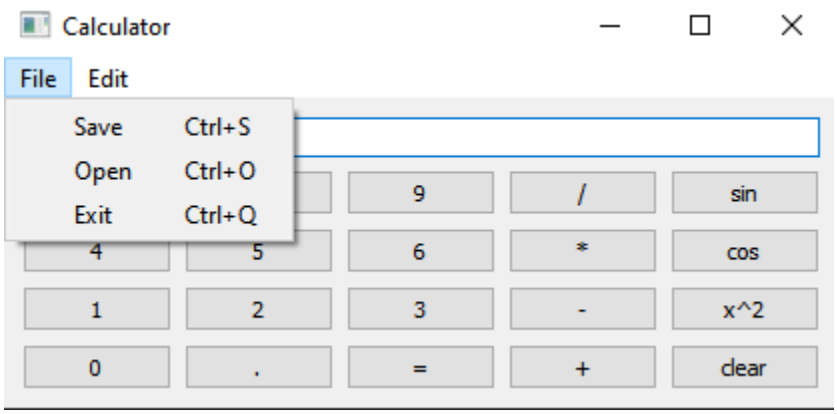
```python
from PyQt5.QtWidgets import *
from PyQt5.QtGui import QIcon
import math

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Calculator")
        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')

        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)

        clearButton = QAction('Clear', self)
        clearButton.setShortcut('Ctrl+C')
        clearButton.triggered.connect(self.cleartext)
        editMenu.addAction(clearButton)

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

    def saveFileDialog(self):
        options = QFileDialog.Options()
        fileName, _ = QFileDialog.getSaveFileName(self, "Save Calculator Data",
"",
                                                  "Text Files (*.txt);; All Files
(*)", options=options)
        if fileName:
```

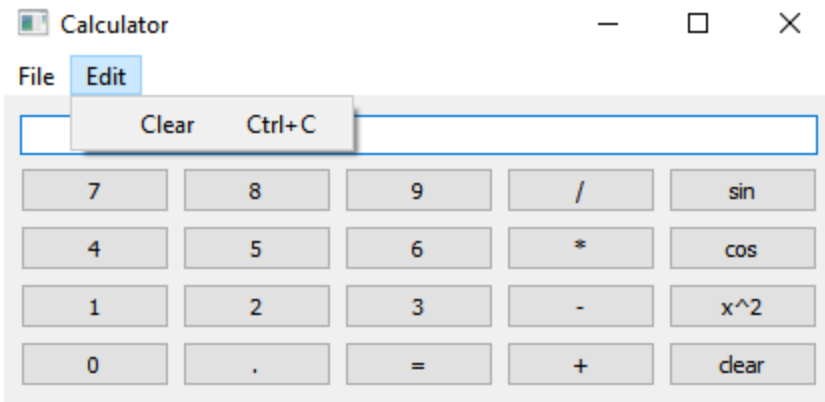
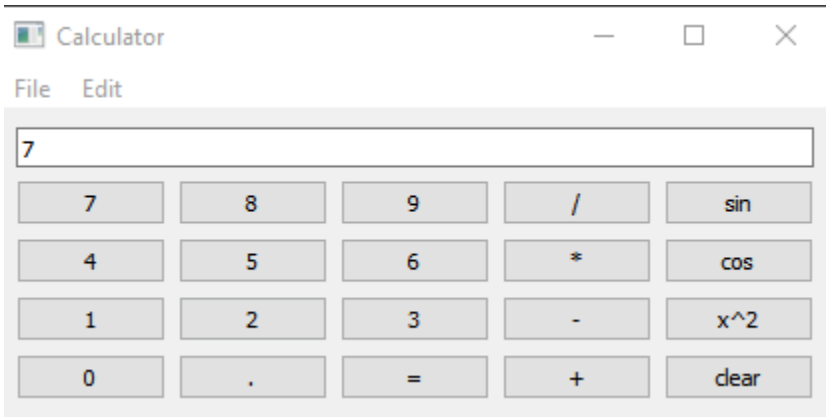
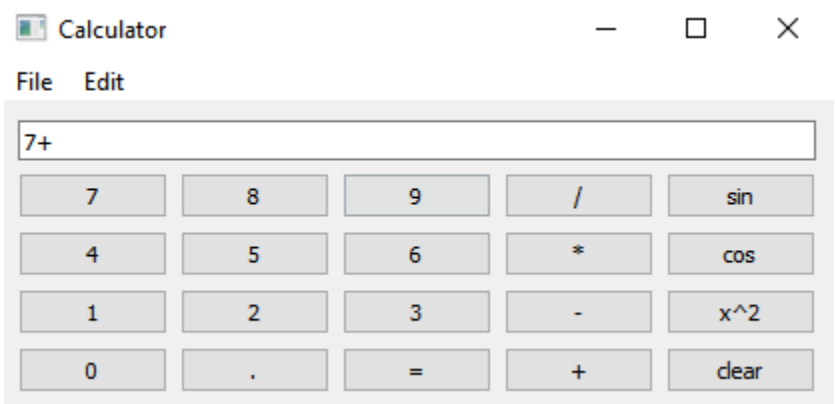
```
            with open(fileName, 'w') as file:
                file.write(self.calculator.display.text())

    def openFileNameDialog(self):
        options = QFileDialog.Options()
        fileName, _ = QFileDialog.getOpenFileName(self, "Open Calculator Data",
"",
                                                  "Text Files (*.txt);; All Files
(*)", options=options)
        if fileName:
            with open(fileName, 'r') as file:
                data = file.read()
                self.calculator.display.setText(data)

    def cleartext(self):
        self.calculator.display.clear()

    def loadwidget(self):
        self.calculator = Calculator()
        self.setCentralWidget(self.calculator)

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

    def initUI(self):
        grid = QGridLayout()
        self.setLayout(grid)

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

        self.display = QLineEdit(self)
        grid.addWidget(self.display, 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)
            button.clicked.connect(lambda checked, name=name:
self.buttonClicked(name))
```

```
        grid.addWidget(button, *position)

    def buttonClicked(self, name):
        if name == '=':
            self.calculate()
        elif name == 'clear':
            self.clearDisplay()
        elif name == 'sin':
            self.sinFunction()
        elif name == 'cos':
            self.cosFunction()
        elif name == 'x^2':
            self.squareFunction()
        elif name == '+':
            self.addFunction()
        elif name == '-':
            self.subtractFunction()
        elif name == '*':
            self.multiplyFunction()
        elif name == '/':
            self.divideFunction()
        else:
            self.appendToDisplay(name)

    def calculate(self):
        try:
            result = str(eval(self.display.text()))
            self.display.setText(result)
        except:
            self.display.setText('Math Error')

    def clearDisplay(self):
        self.display.setText('')

    def sinFunction(self):
        try:
            angle = float(self.display.text())
            result = str(math.sin(math.radians(angle)))
            self.display.setText(result)
        except:
            self.display.setText('Math Error')

    def cosFunction(self):
        try:
            angle = float(self.display.text())
            result = str(math.cos(math.radians(angle)))
            self.display.setText(result)
```

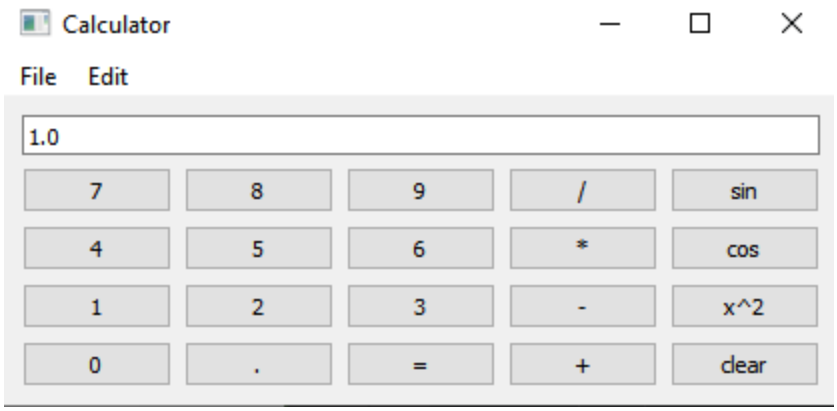
```
        except:
            self.display.setText('Math Error')

    def squareFunction(self):
        try:
            number = float(self.display.text())
            result = str(number ** 2)
            self.display.setText(result)
        except:
            self.display.setText('Math Error')

    def addFunction(self):
        self.appendToDisplay('+')

    def subtractFunction(self):
        self.appendToDisplay('-')

    def multiplyFunction(self):
        self.appendToDisplay('*')

    def divideFunction(self):
        self.appendToDisplay('/')

    def appendToDisplay(self, name):
        self.display.setText(self.display.text() + name)

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

THE OUTPUT

INTERFACE OF THE CALCULATOR	
INTERFACE OF THE CALCULATOR WITH THE SAVE, OPEN, AND EXIT COMMAND	

	
SAMPLE DEMONSTRATION FOR THE ALGEBRAIC OPERATION **flow of the input** **7 -> + -> 3 -> =** **output 10**	 

	 
SAMPLE DEMONSTRATION USING THE TRIGONOMETRIC OPERATION **flow of the input** **90 -> sin** **output 1.0**	



CONCLUSION:

In this activity, we explored more features of PyQt5 by building a calculator and a notepad application. For the supplementary task, I developed a calculator with trigonometric and exponential functions and added commands to save, open, quit, and clear the activity. I also learned to use math.radians which is not part of the exercise but I needed it for the trigonometric functions to work, I found that it is useful when working with trigonometric functions, as it returns results in radians. By modifying some of the original code from the exercises, I gained a better understanding of how common applications like calculators and notepads are built.