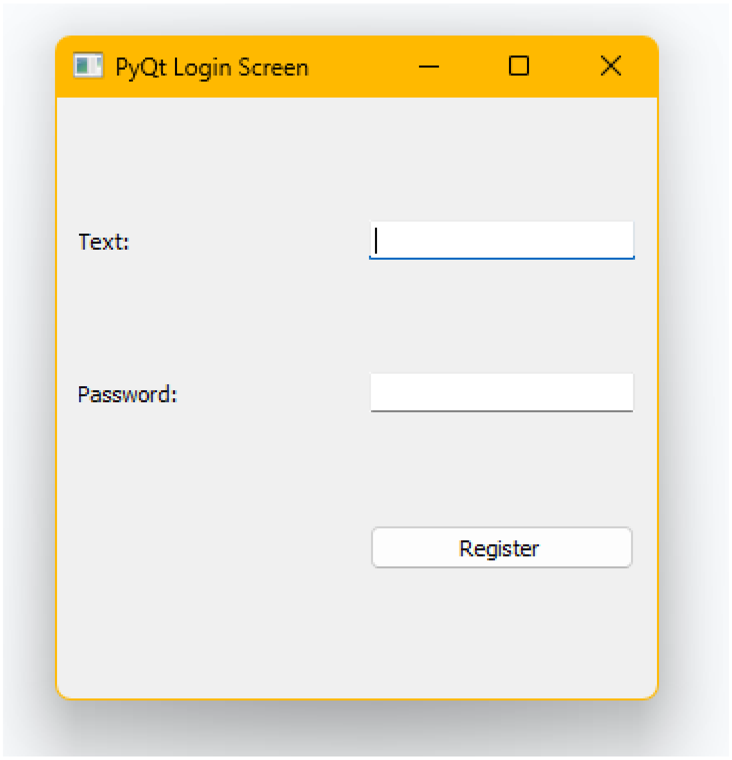
Laboratory Activity 6 - GUI Design: Layout and Styling	
Rio, Aries, C.	10/28/24
BSCPE - CPE21S4	Ma'am MAria Rizette Sayo

Grid Layout

Code:

```
import sys
from PyQt5.QtWidgets import 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_())
```

Output:



Observation:

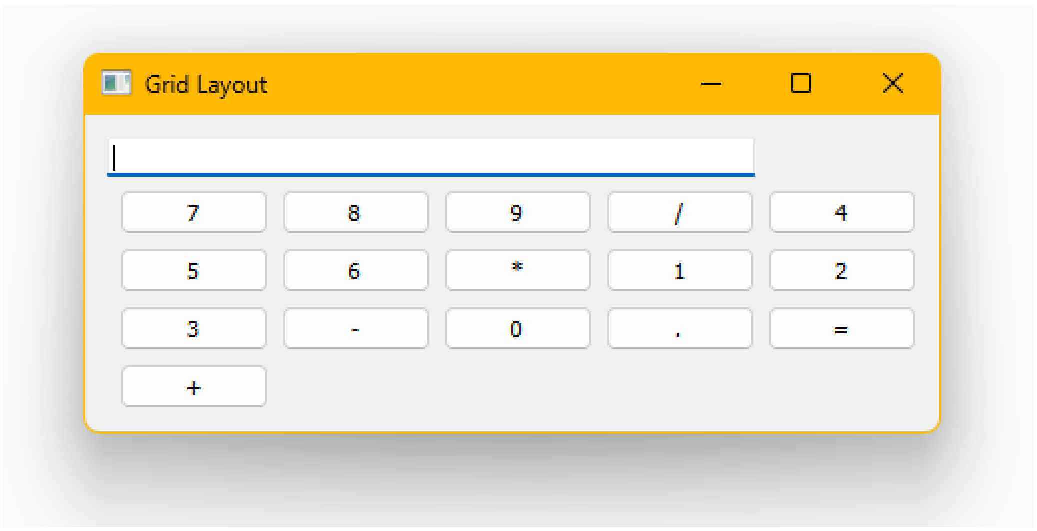
When you run the application, you'll see a neatly organized layout with a "Text: " label and input field aligned horizontally at the top, followed by a "Password: " label and input field directly beneath it. The "Register" button is positioned to the right of the password input field, creating a clean and intuitive user interface for entering credentials. The components should be well-spaced, providing a clear and user-friendly experience.

Grid Layout using Loops

Code:

```
import sys
from  PyQt5.QtWidgets  import  QGridLayout,  QLineEdit,  QPushButton,  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, 0, 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_())
```

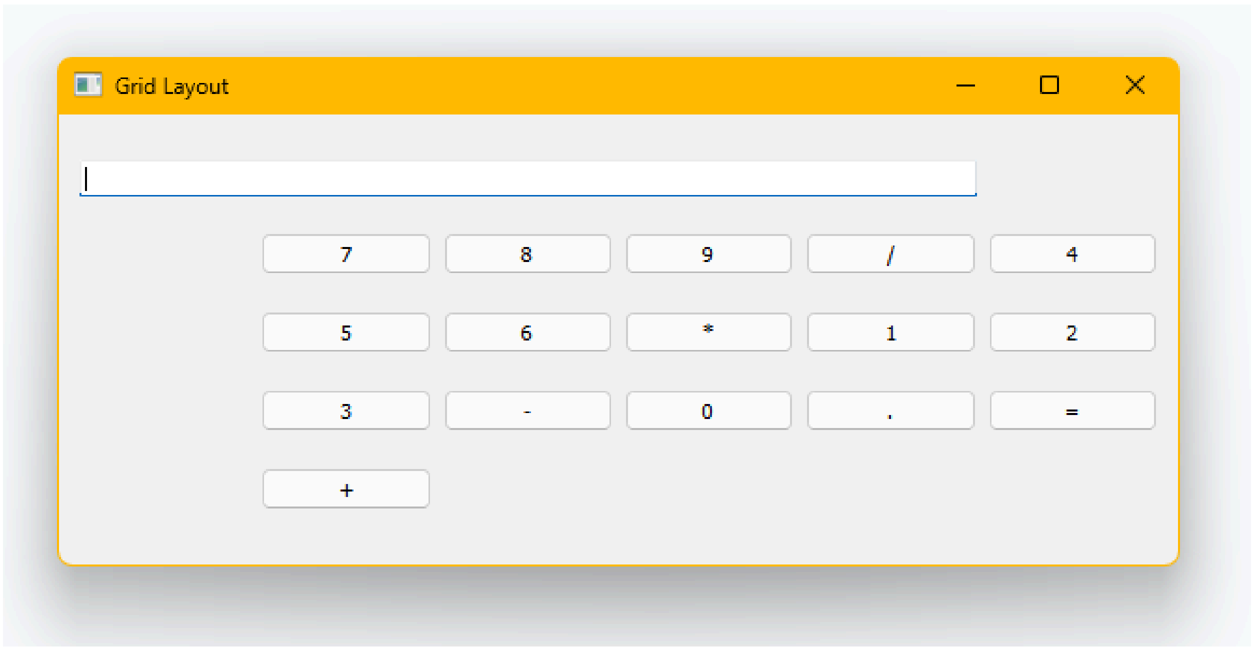
Output:



Observation:

When you run the program, a window titled "Grid Layout" appears, featuring a text input field at the top followed by a grid of buttons. The buttons include numbers 0-9, as well as operation symbols like '+', '-', '*', and '/'. Each button is neatly arranged, making it easy to use as a simple calculator interface. The layout is clean and user-friendly, allowing for straightforward interaction.

Try stretching the window, show the appearance and note your observations:



When you stretch the window, the buttons and text input field adjust while maintaining their grid layout. The text input field expands to fill more space, making it easier to read longer entries, while the buttons remain aligned but may have extra space around them, enhancing the overall appearance and usability.

Vbox and Hbox layout managers (Simple Notepad)

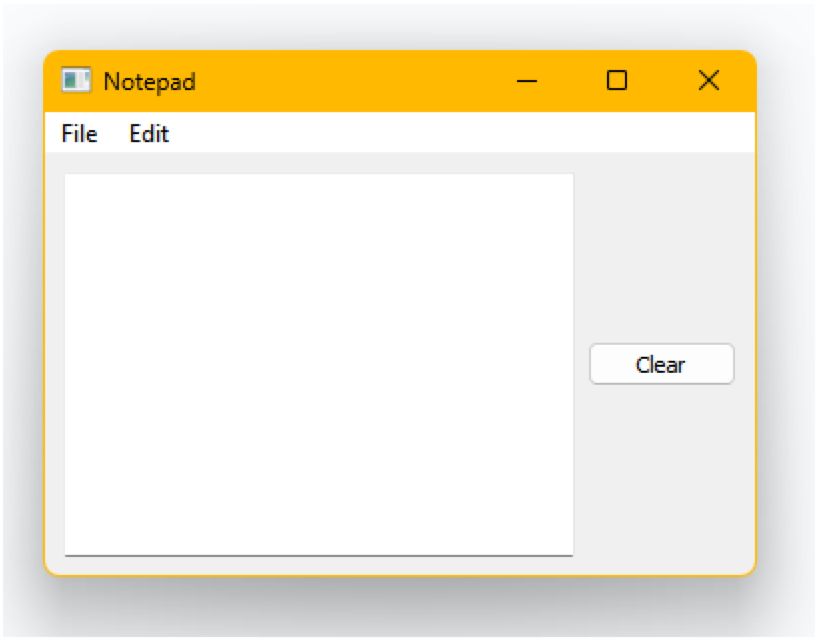
Code:

```
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()
```

```
        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(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.setLayout(windowLayout)

    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_())
```

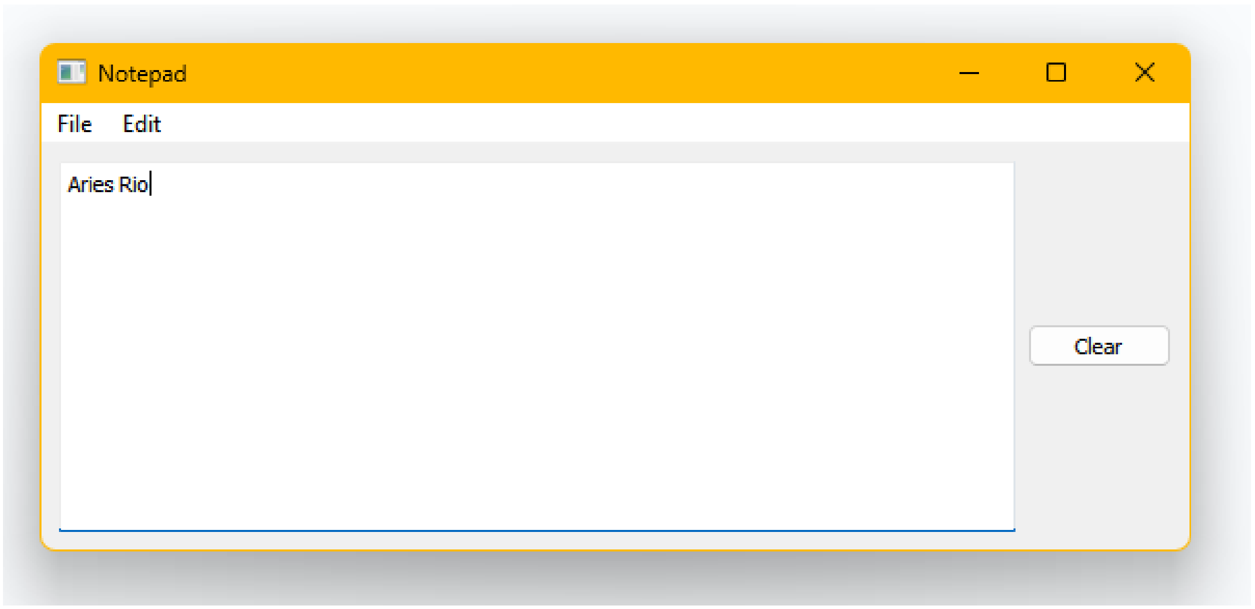
Output:



Observation:

The GUI shows a simple Notepad interface. The menu bar includes "File" and "Edit" options for opening, saving, and clearing text. The text area is spacious for typing, and the "Clear" button works to remove text.

Try to stretch the window and take note of the response of the GUI:



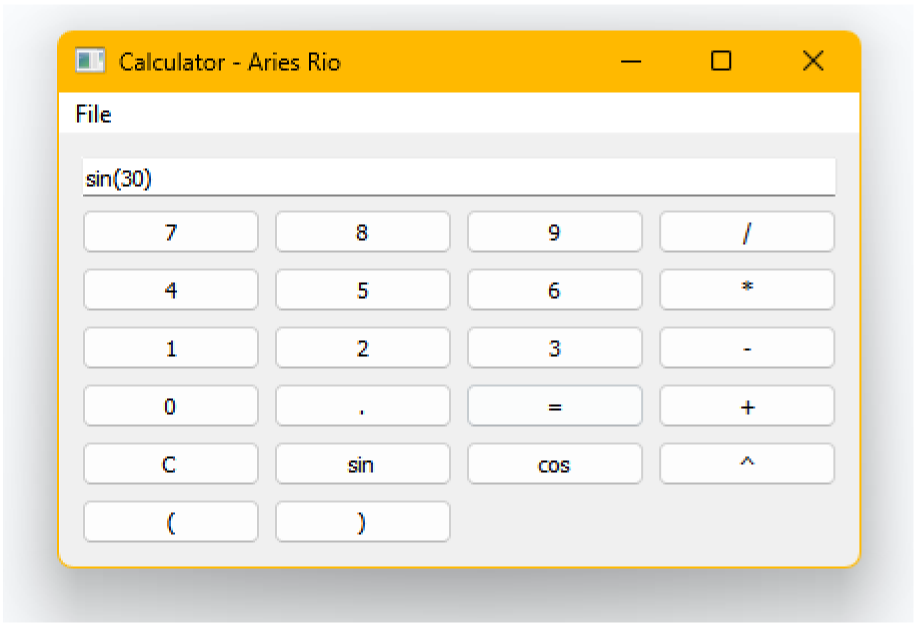
When you stretch the window, the text area expands to fill the available space. The clear button stays in place, and the overall proportions of the GUI remain balanced. This responsiveness allows for comfortable editing, making it easy to view and manipulate text as the window size changes.

Supplementary Activity:

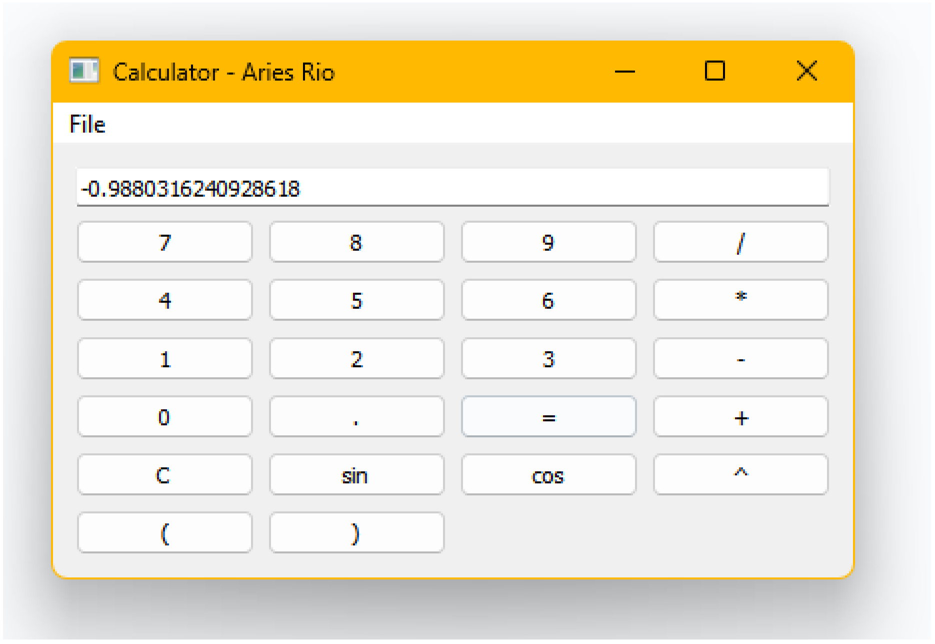
Code:

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



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

    def initUI(self):
        self.setGeometry(300, 300, 400, 300)
        self.setWindowTitle('Advanced Calculator')
        self.textLine = QLineEdit(self)

        grid = QGridLayout()
        grid.addWidget(self.textLine, 0, 0, 1, 4)

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

        # Create buttons and add to grid layout
        positions = [(i, j) for i in range(1, 6) for j in range(4)]
        for position, name in zip(positions, names):
            button = QPushButton(name)
            grid.addWidget(button, *position)
            button.clicked.connect(self.on_button_click)

        # Create menu bar
        menubar = QMenuBar(self)
        file_menu = menubar.addMenu('File')
        save_action = QAction('Save', self)
        save_action.triggered.connect(self.save_to_file)
        exit_action = QAction('Exit - Ctrl Q', self)
        exit_action.triggered.connect(self.close)

        file_menu.addAction(save_action)
        file_menu.addAction(exit_action)

        # Add menu bar to the layout
        vbox = QVBoxLayout()
        vbox.setMenuBar(menubar)
        vbox.addLayout(grid)
        self.setLayout(vbox)

        self.show()

    def on_button_click(self):
        sender = self.sender()
        text = sender.text()

        if text == '=':
            try:
                # Replace '^' with '**' for exponentiation
                expression = self.textLine.text().replace('^', '**')
                result = eval(expression)
                self.textLine.setText(str(result))
                self.save_operation(expression, result)
            except Exception:
                self.textLine.setText('Error')
        elif text == 'C':
            self.textLine.clear()
        else:
            # Append the button text to the line edit
            current_text = self.textLine.text()
            self.textLine.setText(current_text + text)

    def save_operation(self, operation, result):
        with open('calculations.txt', 'a') as f:
            f.write(f"{operation} = {result}\n")

    def save_to_file(self):
        options = QFileDialog.Options()
         file_name, _ = QFileDialog.getSaveFileName(self, "Save File", "", "Text
Files (*.txt);;All Files (*)",
                                                  options=options)
        if file_name:
            with open(file_name, 'w') as f:
                f.write("Calculation History:\n")

                with open('calculations.txt', 'r') as history:

                    f.write(history.read())


    def closeEvent(self, event):

          reply = QMessageBox.question(self, 'Exit Confirmation', 'Are you sure
 you want to exit?',

                                                 QMessageBox.Yes | QMessageBox.No,
 QMessageBox.No)

        if reply == QMessageBox.Yes:

            event.accept()

        else:

            event.ignore()




 if __name__ == '__main__':

    app = QApplication(sys.argv)

    ex = Calculator()

    sys.exit(app.exec_())
```

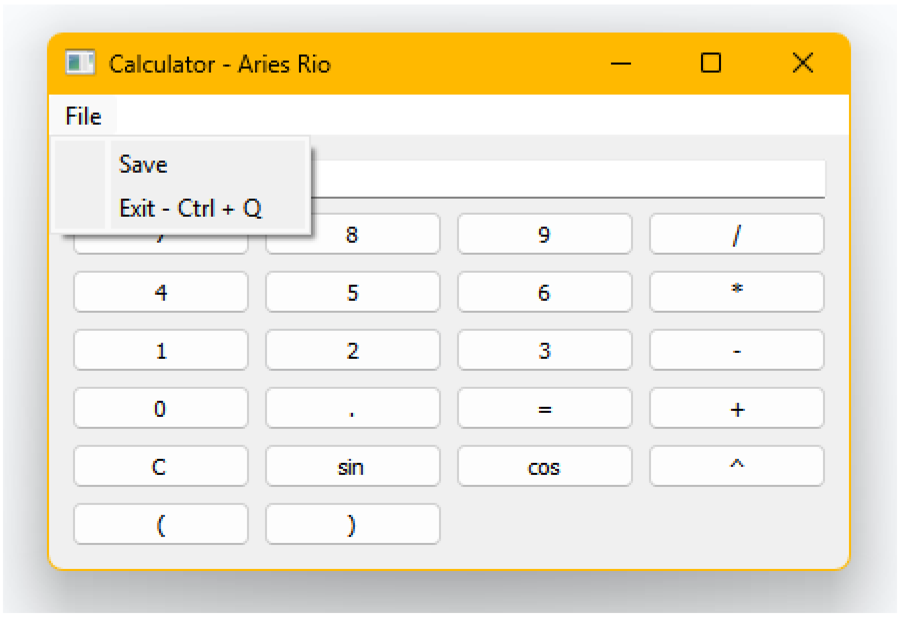
Output:

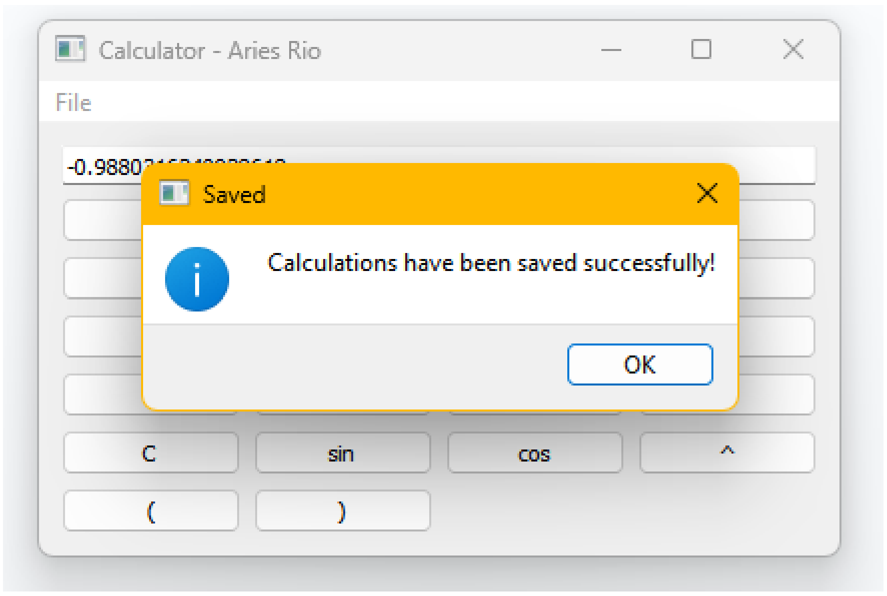


```
sin(30)
```

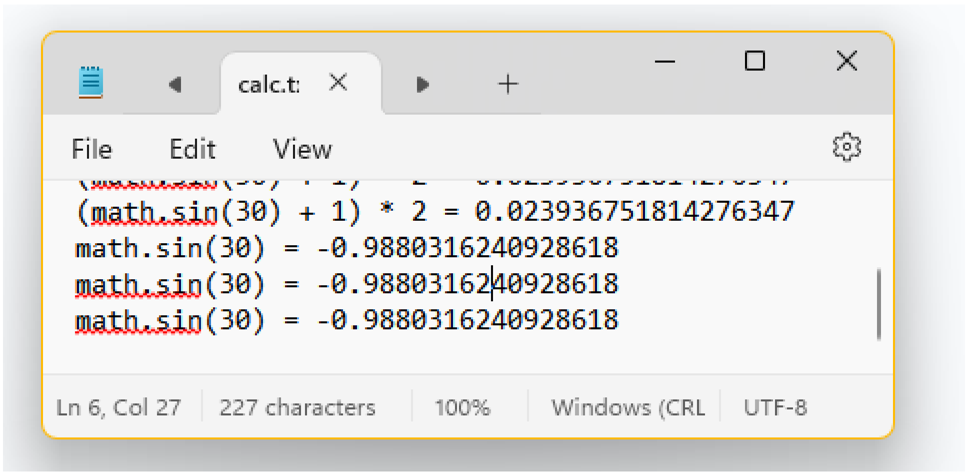


```
-0.9880316240928618
```

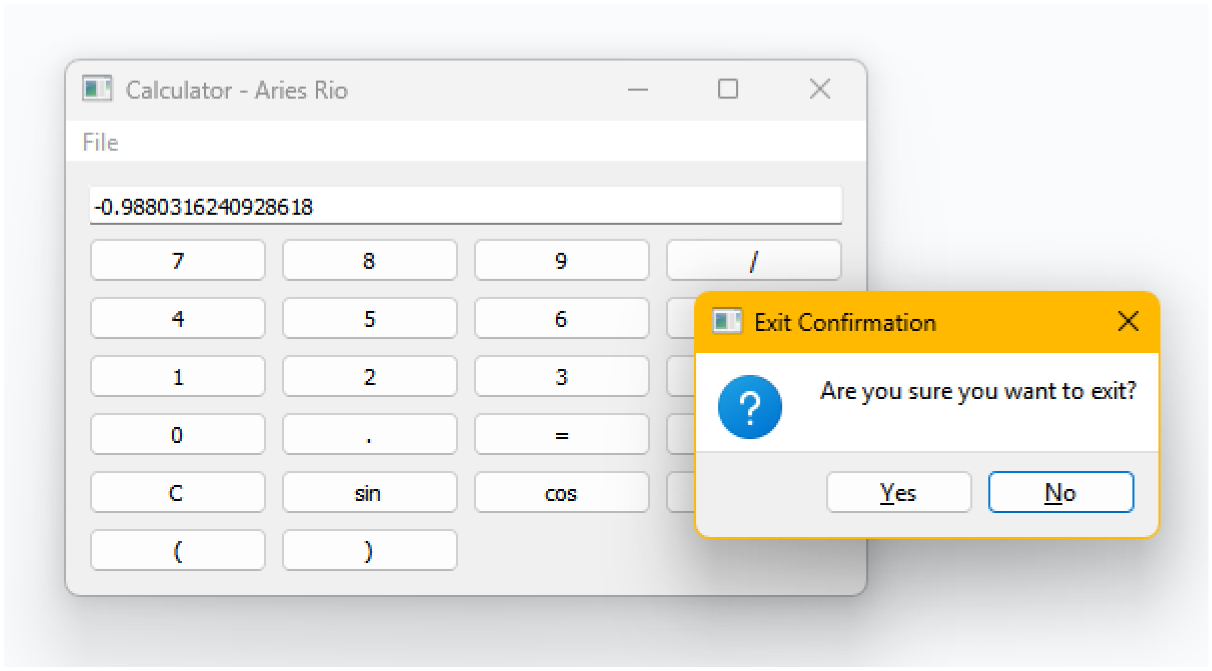




Calculations have been saved successfully!



```
(math.sin(30) + 1) * 2 = 0.023936751814276347
math.sin(30) = -0.9880316240928618
math.sin(30) = -0.9880316240928618
math.sin(30) = -0.9880316240928618
```



```
-0.9880316240928618
```

Are you sure you want to exit?

Conclusion:

I learned the basics of GUI layout management using grid, VBox, and HBox layouts in Python. By building different GUI applications, I practiced positioning components and seeing how they adjust when the window is resized. Creating a calculator for the supplementary activity helped me understand event handling and file operations better.