Laboratory Activity No. 6	
GUI Design: Layout and Styling	
Course Code: CPE009	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to familiarize students on how to implement layout and styling in a GUI Application.

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Identify the layout managers that can be used in PyQt5
2.2 Create a GUI program with layout and stylesheets.

3. Discussion:

A Graphical User Interface (GUI) application manages its components the position of its components using two ways. One is using Absolute Positioning (x,y coordinates from the top-left of the screen which is 0,0). The second is by using Layout Managers. There are various layout managers available in PyQt5. This lab activity will explore using the different layout managers available to organize PyQt5 Widgets.

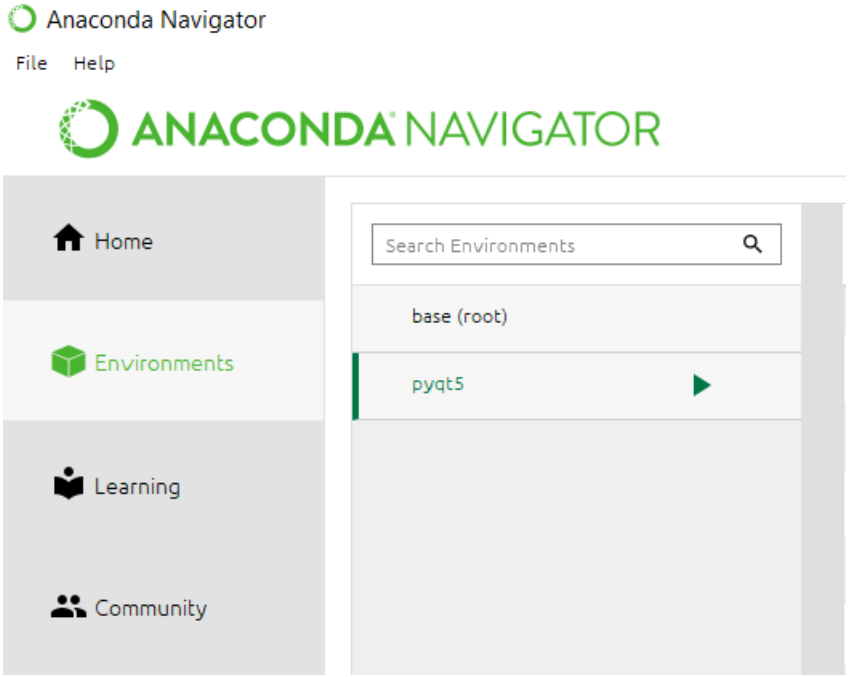
4. Materials and Equipment:

Desktop Computer with Anaconda Python
Windows Operating System

5. Procedure:

Using the pyqt5 Virtual Environment

1. Open the Anaconda Navigator and go to Environments
2. Select the pyqt5 virtual environment that you created.



Basic Grid Layout

1. Create a folder named oopfa1<lastname>_lab11
2. Open your Anaconda Navigator and select Visual Studio Code or Spyder IDE.
3. Open that folder in your editor and create a file named **gui_grid1.py** then copy the code as shown:

```
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_())
```

4. Run the program and observe the positioning of the components.

Grid Layout using Loops

1. Create a new file named **gui_grid2.py** and copy and run the following code:

```python
#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)

        # 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_())
```

2. Run the program and observe the output.
3. Try stretching the window, show the appearance and note your observations.

Vbox and Hbox layout managers (Simple Notepad)

1. Create a new file named **gui_simplenotepad.py** and copy the following code below:

MainWindow Class

```
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()
        # 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()
```

Code to run the GUI

```
101
102 if __name__ == '__main__':
103     app = QApplication(sys.argv)
104     ex = MainWindow()
105     sys.exit(app.exec_())
```

2. Run the program and observe the output GUI.
3. Try to stretch the window and take note of the response of the GUI.

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.

7. Conclusion:

8. Assessment Rubric: