COSC 6344 Visualization - PyQt Skeleton Code

I. Introduction

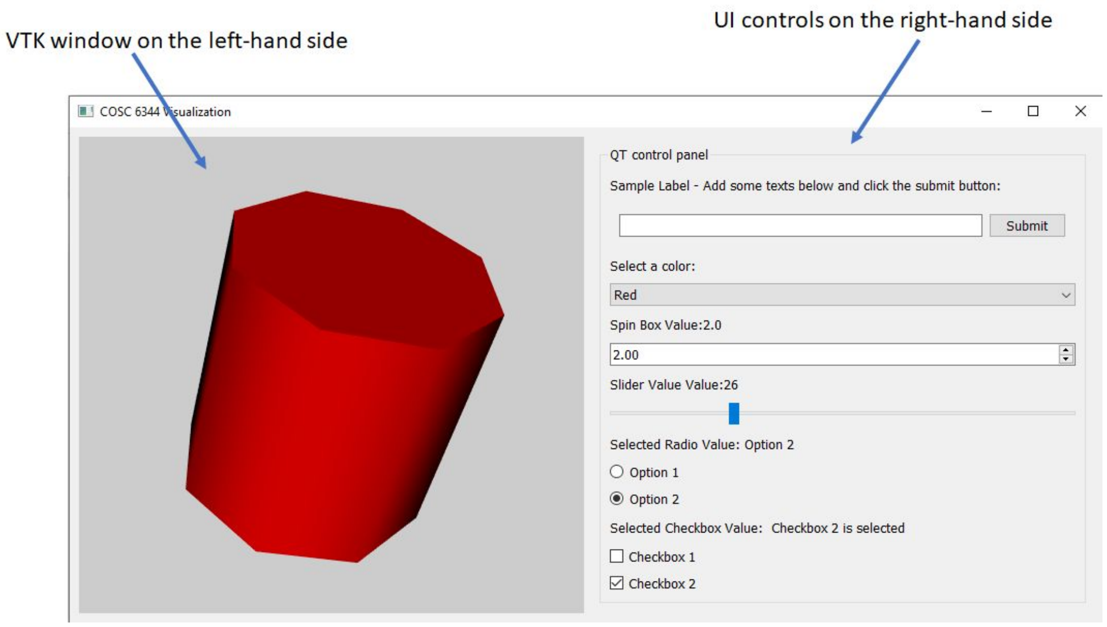
PyQT is a python-based cross-platform GUI library. It provides a large number of classes and functions to create graphical applications. This documentation is not designed to teach everything about PyQT. Instead, it aims to provide tutorials for using basic UI components and focuses on explaining the skeleton code which combines PyQT and PyVTK to create a GUI visualization system. After finishing the documentation, you should be able to extend the skeleton code to design your own GUI applications for the future assignments in the COSC 6344 Visualization class. I highly recommend you to check out the useful tutorials at https://www.tutorialspoint.com/pyqt/index.htm and the official website https://doc.qt.io/qtforpython/index.html to get a better understanding about the library.

II. Skeleton Code

I assume that you have already installed PyVTK and PyQT by following the instructions provided in the class. You can run the skeleton code by simply typing:

```
python qtvtk_skeleton.py
```

Here is the expected interface of our GUI skeleton program:



Sample Label - Add some texts below and click the submit button:

Select a color: Red

Spin Box Value:2.0

2.00

Slider Value Value:26

Selected Radio Value: Option 2

Selected Checkbox Value: Checkbox 2 is selected

Figure 1: The interface of the skeleton program

The skeleton program visualizes a vtk cylindrical model on the left-hand side window and shows basic UI controls in QT on the right-hand side. The UI controls include labels, textbox, button, combo box, spin box, slider, radio button, checkbox. The container for the control is called a widget. You will need to update (e.g add, remove, change) the functions of the controls so that users can interact and generate the desired visualization as described in the assignment. The placeholder functions for the UI controls are provided in the skeleton code to help you get started quickly. For instance, you can change the color of the cylinder via the combo box:

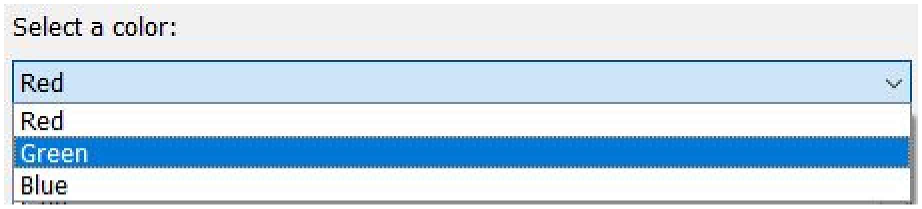
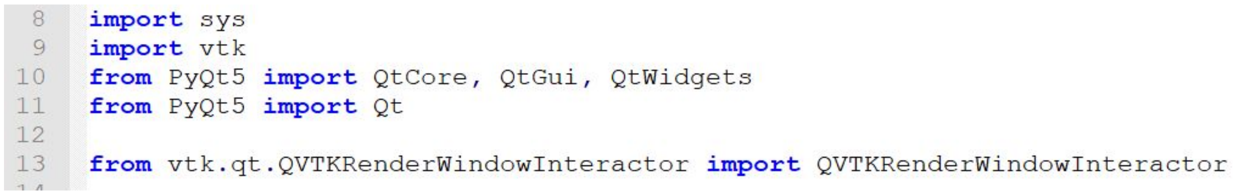


Figure 2: A combo box for the color selection

Creating the skeleton GUI application using PyQt and PyVTK involves the following steps:

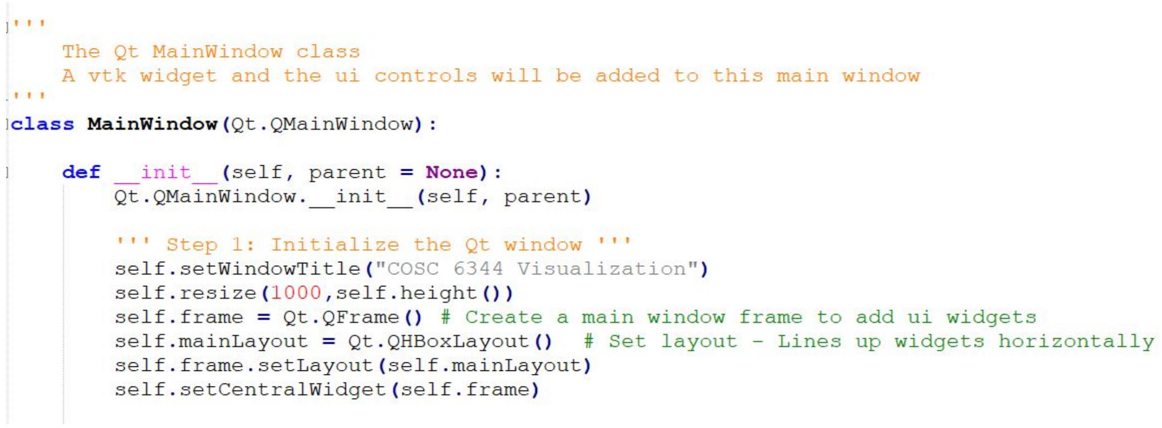
Step 1: Import PyQT and vtk modules



```
import sys
import vtk
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5 import Qt

from vtk.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor
```

Step 2: Create Qt MainWindow class with a central widget



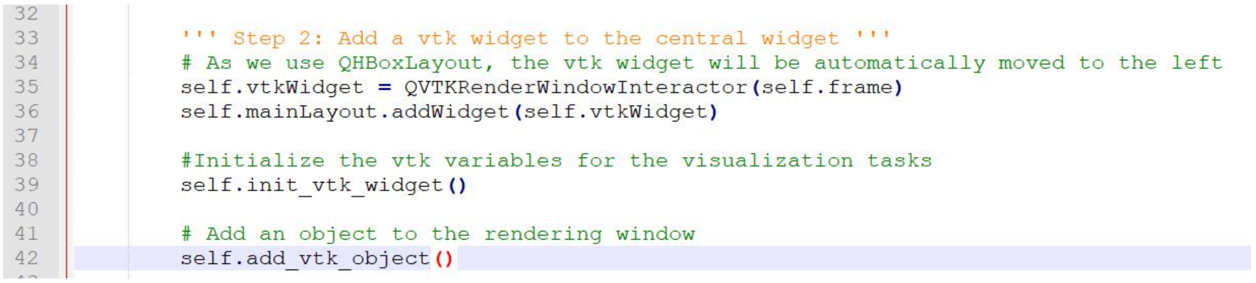
```
'''
    The Qt MainWindow class
    A vtk widget and the ui controls will be added to this main window
'''
class MainWindow(Qt.QMainWindow):

    def __init__(self, parent = None):
        Qt.QMainWindow.__init__(self, parent)

        ''' Step 1: Initialize the Qt window '''
        self.setWindowTitle("COSC 6344 Visualization")
        self.resize(1000,self.height())
        self.frame = Qt.QFrame() # Create a main window frame to add ui widgets
        self.mainLayout = Qt.QHBoxLayout()  # Set layout - Lines up widgets horizontally
        self.frame.setLayout(self.mainLayout)
        self.setCentralWidget(self.frame)
```

Note that QT widgets need a layout that helps to place the nested UI controls/widgets in a proper position. The two most popular layouts are QHBoxLayout and QVBoxLayout. The former lines up widgets horizontally and the latter places widgets vertically. The main layout for our skeleton code is QHBoxLayout.

Step 3: Add VTK widgets (or VTK window) to the central widget



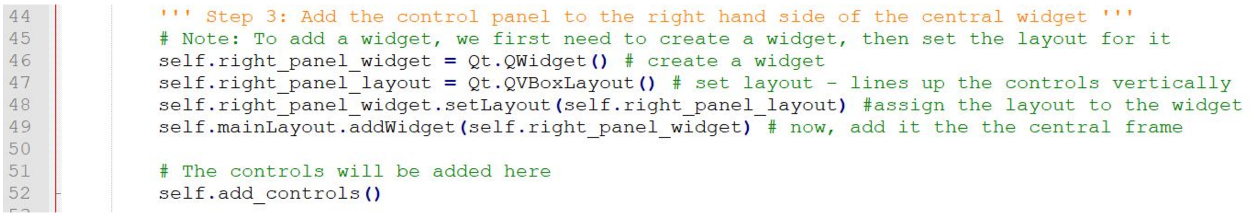
```
        ''' Step 2: Add a vtk widget to the central widget '''
        # As we use QHBoxLayout, the vtk widget will be automatically moved to the left
        self.vtkWidget = QVTKRenderWindowInteractor(self.frame)
        self.mainLayout.addWidget(self.vtkWidget)

        #Initialize the vtk variables for the visualization tasks
        self.init_vtk_widget()

        # Add an object to the rendering window
        self.add_vtk_object()
```

As we use QHBoxLayout, the vtk widget is added to the left-hand side automatically. You can check out the functions ***init_vtk_widget()*** and ***add_vtk_object()*** to see how to add vtk codes to the program.

Step 4: Add QT UI controls to the control panel on the right-hand side and create the event handling functions

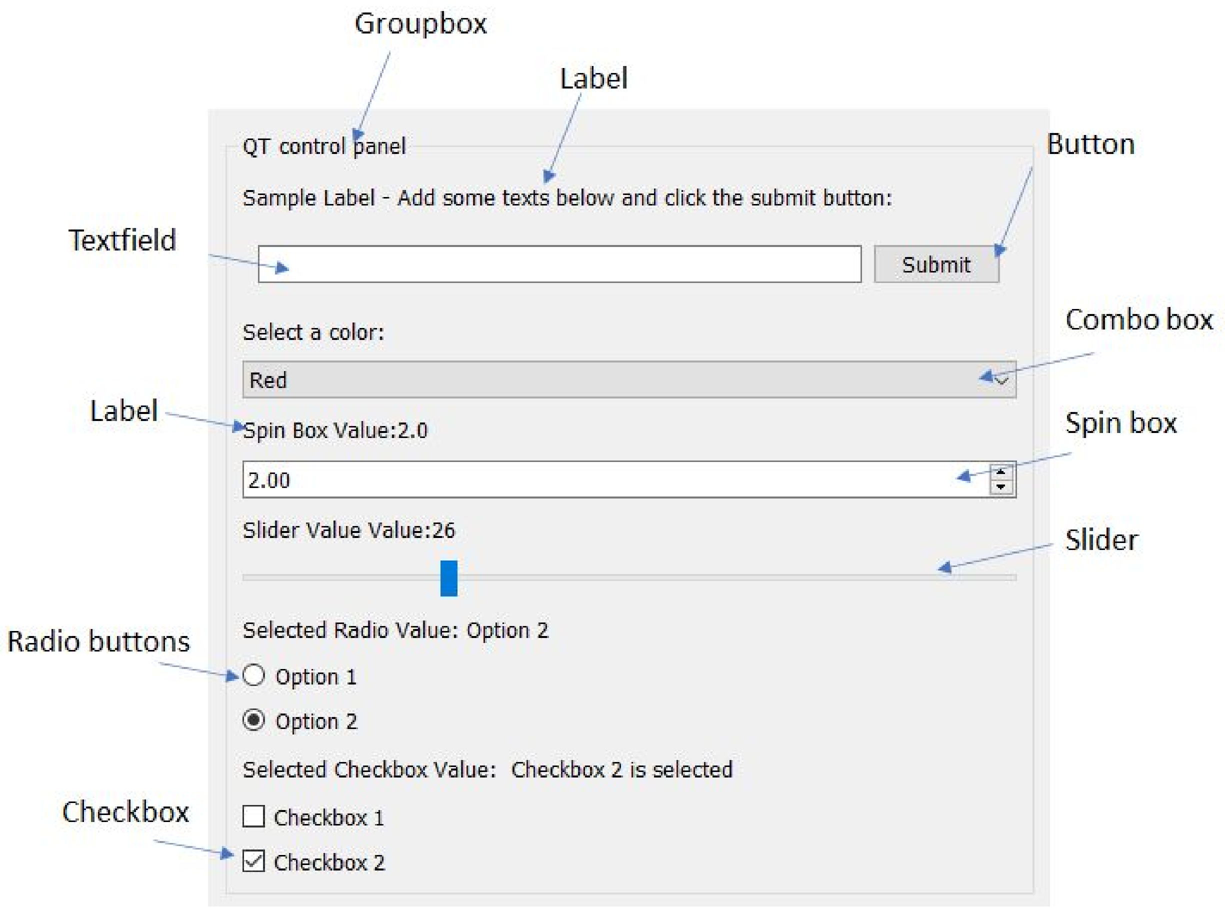


```
        ''' Step 3: Add the control panel to the right hand side of the central widget '''
        # Note: To add a widget, we first need to create a widget, then set the layout for it
        self.right_panel_widget = Qt.QWidget() # create a widget
        self.right_panel_layout = Qt.QVBoxLayout() # set layout - lines up the controls vertically
        self.right_panel_widget.setLayout(self.right_panel_layout) #assign the layout to the widget
        self.mainLayout.addWidget(self.right_panel_widget) # now, add it the the central frame

        # The controls will be added here
        self.add_controls()
```

The control panel is a QT Widget which is the container for the ui control. It uses QVBoxLayout which means the ui controls will be placed vertically. In the next section, we will go deeper to the function **add_controls()** to see how we can add QT UI controls and set their event handling functions.

III. Adding QT UI controls and setting the event handling functions



QT control panel

Sample Label - Add some texts below and click the submit button:

Submit

Select a color:

Red

Spin Box Value:2.0

2.00

Slider Value Value:26

Selected Radio Value: Option 2

Option 1

Option 2

Selected Checkbox Value: Checkbox 2 is selected

Checkbox 1

Checkbox 2

Figure 3: UI controls in the control panel

1. **Groupbox**
   The skeleton code uses a groupbox (QGroupBox) to group all controls in one box with a cleared rectangle boundary.

```
    '''
        Add QT controls to the control panel in the righ hand size
    '''
    def add_controls(self):

        ''' Add a sample group box '''
        groupBox = Qt.QGroupBox("QT control panel") # Use a group box to group controls
        groupBox_layout = Qt.QVBoxLayout() #lines up the controls vertically
        groupBox.setLayout(groupBox_layout)
        self.right_panel_layout.addWidget(groupBox)
```

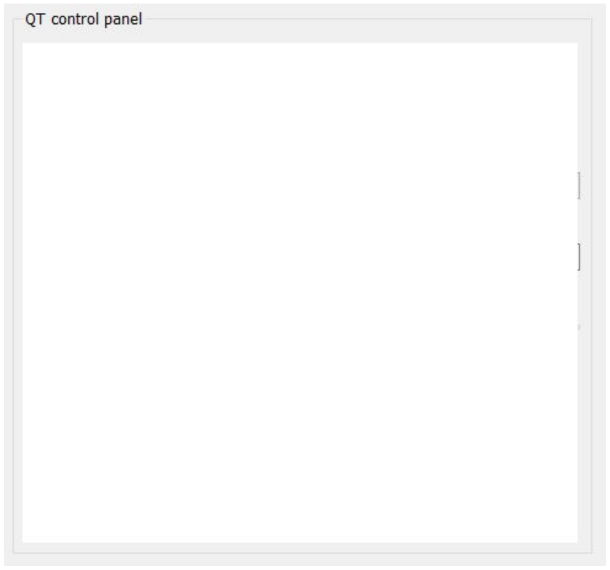


Figure 4: Groupbox with the title “QT control panel”

2. **Label**
   This is a simple way to add a label by using QLabel

```
''' This is a sample label '''
label = Qt.QLabel("Sample Label - Add some texts below and click the submit button:")
groupBox_layout.addWidget(label)
```

   To update the text of the label, please use the **.setText()** function

3. **Text field**
   This is a simple way to add a text field by using QLineEdit

```
self.qt_textfield = Qt.QLineEdit() # create text field
hbox.addWidget(self.qt_textfield)
```

   To get the text value, just use **.text()**

4. **Button**
   This is a simple way to add a button by using QPushButton

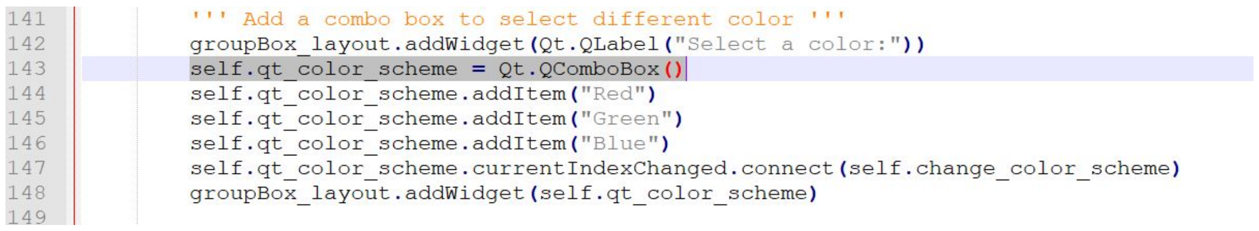
```
self.qt_submit_button = Qt.QPushButton('Submit') #create a button
self.qt_submit_button.clicked.connect(self.on_submit_clicked)
self.qt_submit_button.show()
hbox.addWidget(self.qt_submit_button)
```

   Here, we set the function **on_submit_clicked()** to handle the click event.

```
''' Handle the click event for the submit button  '''
def on_submit_clicked(self):
    self.show_popup_message(self.qt_textfield.text())
```

   It will basically popup a message that users type in the neighboring text field.

5. **Combo Box**
   This is a simple way to add a combo box by using QComboBox

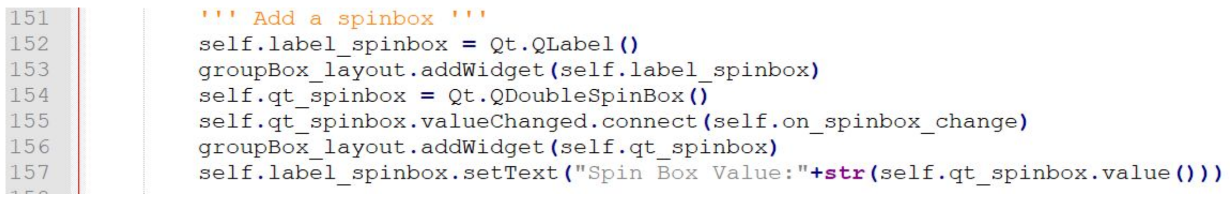


```python
''' Add a combo box to select different color '''
groupBox_layout.addWidget(Qt.QLabel("Select a color:"))
self.qt_color_scheme = Qt.QComboBox()
self.qt_color_scheme.addItem("Red")
self.qt_color_scheme.addItem("Green")
self.qt_color_scheme.addItem("Blue")
self.qt_color_scheme.currentIndexChanged.connect(self.change_color_scheme)
groupBox_layout.addWidget(self.qt_color_scheme)
```

We set the function **change_color_scheme()** to handle the event of the combo box. Note that the label here helps to clarify the meaning of the combo box.

6. **Spin box**

   This is a simple way to add a spin box by using QSpinBox

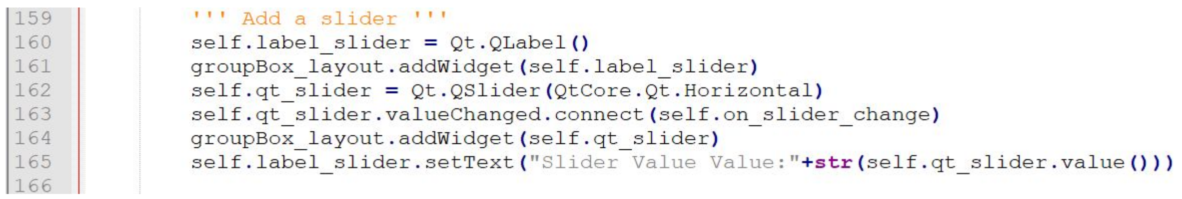


```python
''' Add a spinbox '''
self.label_spinbox = Qt.QLabel()
groupBox_layout.addWidget(self.label_spinbox)
self.qt_spinbox = Qt.QDoubleSpinBox()
self.qt_spinbox.valueChanged.connect(self.on_spinbox_change)
groupBox_layout.addWidget(self.qt_spinbox)
self.label_spinbox.setText("Spin Box Value:"+str(self.qt_spinbox.value()))
```

   We set the function **on_spinbox_change()** to handle the event of the spin box. In the skeleton code, the nearby label will show the value of the spinbox when it changes.

7. **Slider**

   This is a simple way to add a slider by using QSlider

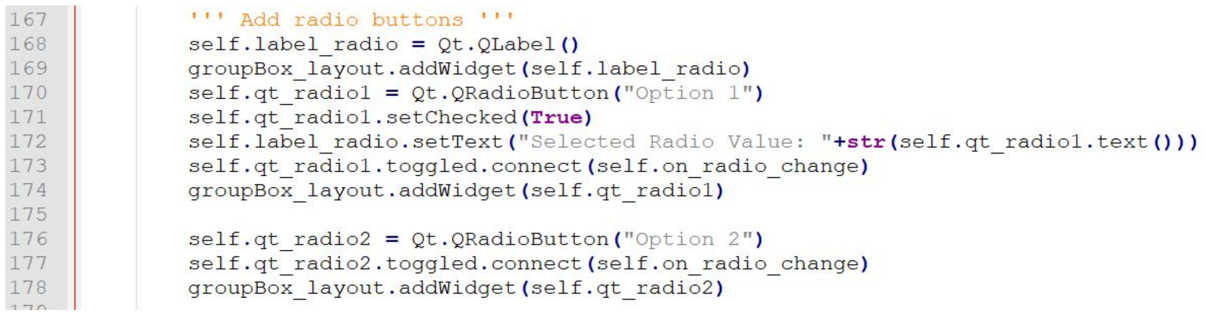


```python
''' Add a slider '''
self.label_slider = Qt.QLabel()
groupBox_layout.addWidget(self.label_slider)
self.qt_slider = Qt.QSlider(QtCore.Qt.Horizontal)
self.qt_slider.valueChanged.connect(self.on_slider_change)
groupBox_layout.addWidget(self.qt_slider)
self.label_slider.setText("Slider Value Value:"+str(self.qt_slider.value()))
```

   We set the function **on_slider_change()** to handle the changes of the slider.

8. **Radio Buttons**

   This is a simple way to add two radio buttons by using QRadioButton



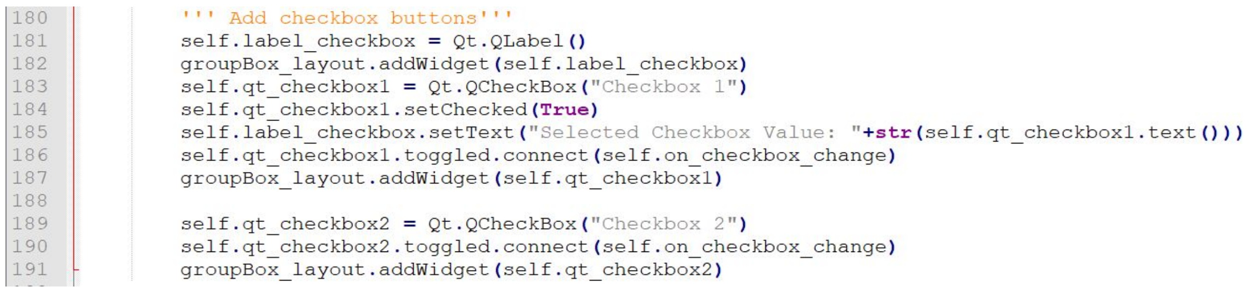
```python
''' Add radio buttons '''
self.label_radio = Qt.QLabel()
groupBox_layout.addWidget(self.label_radio)
self.qt_radio1 = Qt.QRadioButton("Option 1")
self.qt_radio1.setChecked(True)
self.label_radio.setText("Selected Radio Value: "+str(self.qt_radio1.text()))
self.qt_radio1.toggled.connect(self.on_radio_change)
groupBox_layout.addWidget(self.qt_radio1)

self.qt_radio2 = Qt.QRadioButton("Option 2")
self.qt_radio2.toggled.connect(self.on_radio_change)
groupBox_layout.addWidget(self.qt_radio2)
```

   We set the function **on_radio_change()** to handle the changes of the radio buttons

9. **Checkbox**

   This is a simple way to add two checkboxes by using QCheckBox



```
        ''' Add checkbox buttons'''
        self.label_checkbox = Qt.QLabel()
        groupBox_layout.addWidget(self.label_checkbox)
        self.qt_checkbox1 = Qt.QCheckBox("Checkbox 1")
        self.qt_checkbox1.setChecked(True)
        self.label_checkbox.setText("Selected Checkbox Value: "+str(self.qt_checkbox1.text()))
        self.qt_checkbox1.toggled.connect(self.on_checkbox_change)
        groupBox_layout.addWidget(self.qt_checkbox1)

        self.qt_checkbox2 = Qt.QCheckBox("Checkbox 2")
        self.qt_checkbox2.toggled.connect(self.on_checkbox_change)
        groupBox_layout.addWidget(self.qt_checkbox2)
```

We set the function **on_checkbox_change()** to handle the changes of the checkboxes. Last but not least, the comments in the code might help to explain the meaning of variables and functions. If you found any issues, please report it to the instructor.