2019

PyQt5 Tutorial

A SAMPLE APPLICATION

LILIAN SAO DE RIVERA

Contents

List of Figures 2
Code for each Section 2
List of Class Diagrams 2
Basic PyQt5 Application 3
No.1 Creating a Menu and a Menu Item. 3
No.2 Creating a second option and print a message box upon request 5
No.3 Managing different types of Layouts 8
No.4 Using controls in PyQT5: checkbox 15
No.5 Using controls in PyQT5: LineEdit and Pushbutton 19
No.6 Using controls in PyQT5: Radio Buttons 24
No.7 Putting everything together: GroupBox, MatplotLib , and a Graphic with parameters. 31
First Option: GroupBox 33
Second Option: Graphic 34
Third Option: Graphic with Parameters. 34

List of Figures

Figure 1. Menus ____ 3
Figure 2. Hello World Message. ____ 6
Figure 3. Layout Examples ____ 9
Figure 4. Checkbox Example ____ 15
Figure 5. Line Edit and Button Example ____ 19
Figure 6. Radio Button Example ____ 25
Figure 7. GroupBox and Graphics ____ 33

Code for each Section

Table 1. Code for a Simple Menu ____ 3
Table 2. Code for Printing a “Hello World” Message ____ 6
Table 3. Code for Different Layouts ____ 9
Table 4. Code for a Checkbox ____ 15
Table 5. Code for a LIneEdit and a PushButton ____ 20
Table 6. Code for RadioButtons ____ 26
Table 7. Code for a Grupo Box, a Graphic and a Graph with Parameters ____ 36

List of Class Diagrams

Diagram 1. Classes and Methods for Section one ____ 5
Diagram 2. Classes and Methods for Section Two ____ 8
Diagram 3. Classes and Methods for Section Three ____ 14
Diagram 4. Classes and Methods for Section Four ____ 18
Diagram 5. Classes and Methods for Section Five ____ 24
Diagram 6. Classes and Methods for Section Six ____ 31
Diagram 7. Classes and Methods for Section Seven ____ 43

Basic PyQt5 Application

No.1 Creating a Menu and a Menu Item.

The figure below shows a basic window in PyQt5 that shows a Menu bar with one option, a Menu Item with four characteristics, an icon (door), a label (“Exit”) a short key (“Ctrl-Q”) and as short status bar description(“Exit application”). Each time the user hovers over the item menu the status bar shows the description of the action to be taken.

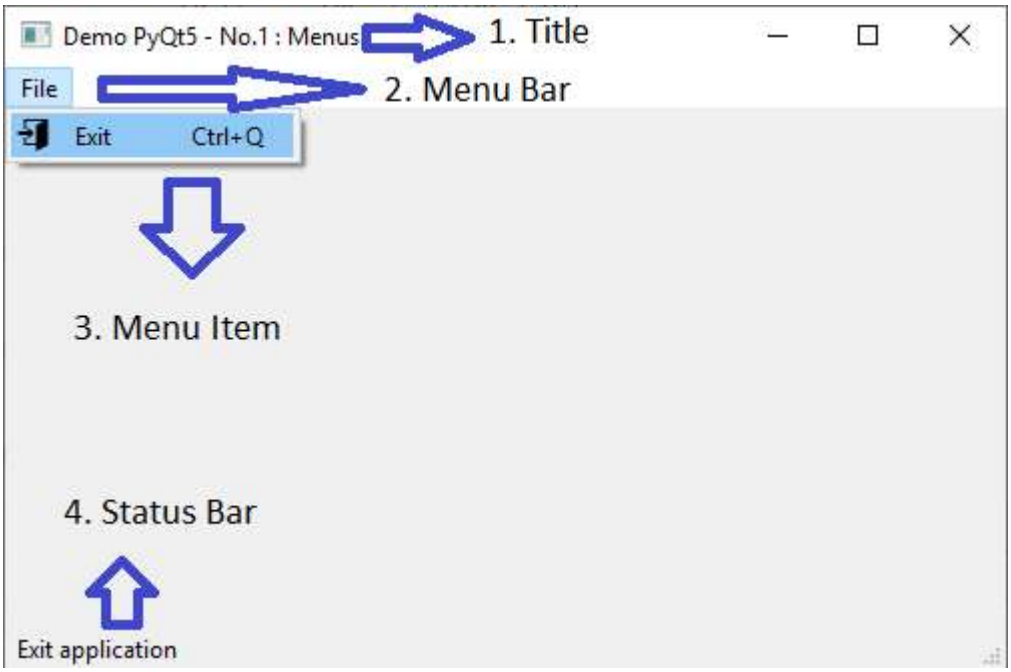


Figure 1. Menus

The following is the code that produces this application. All the explanations for the code are in the form of comments.

Table 1. Code for a Simple Menu

Download the code from :
https://github.com/saoderivera/PyQt5Tutorial/blob/master/TutorialPyQt501.py

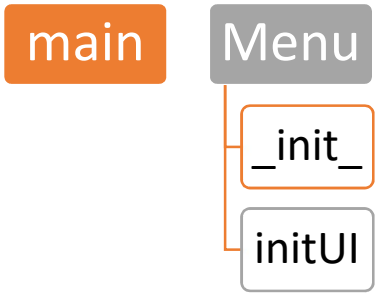
```
#::----------------------------------------------------------------
#:: To create a Menu with options this are the libraries and components that
#:: requiered. For each new option we will be o adding new components
#::----------------------------------------------------------------
import sys
from PyQt5.QtWidgets import QMainWindow, QAction, QMenu, QApplication
from PyQt5.QtGui import QIcon
```

```
#::------------------------------------------------------------
#:: Definition of a Class for the main manu in the application
#::------------------------------------------------------------
class Menu(QMainWindow):

    def __init__(self):

        super().__init__()
        #::-----------------------
        #:: variables use to set the size of the window that contains the menu
        #::-----------------------
        self.left = 100
        self.top = 100
        self.width = 500
        self.height = 300

        #:: Title for the application

        self.Title = 'Demo PyQt5 - No.1 : Menus'

        #:: The initUi is call to create all the necessary elements for the menu

        self.initUI()

    def initUI(self):

        #::-------------------------------------------------
        # Creates the manu and the items
        #::-------------------------------------------------
        self.setWindowTitle(self.Title)
        self.setGeometry(self.left, self.top, self.width, self.height)
        self.statusBar()
        #::-----------------------------
        # 1. Create the menu bar
        # 2. Create an item in the menu bar
        # 3. Creaate an action to be executed when the option
        #    in the  menu bar is choosen
        #::-----------------------------
        mainMenu = self.menuBar()
        fileMenu = mainMenu.addMenu('File')

        #::--------------------------------------
        # Exit action
        # The following code creates the the da Exit Action along
        # with all the characteristics associated with the action
        # The Icon, a shortcut , the status tip that would appear in the window
        # and the action
        #  triggered.connect will indicate what is to be done when the item in
        # the menu is selected
        # These definitions are not available until the button is assigned
        # to the menu
        #::--------------------------------------

        exitButton = QAction(QIcon('enter.png'), '&Exit', self)
        exitButton.setShortcut('Ctrl+Q')
        exitButton.setStatusTip('Exit application')
        exitButton.triggered.connect(self.close)
```

```
        #:: This line adds the button (item element ) to the menu

        fileMenu.addAction(exitButton)

        #:: This line shows the windows

        self.show()

#::------------------------
#:: Application starts here
#::------------------------
def main():
    app = QApplication(sys.argv)  # creates the PyQt5 application
    mn = Menu()  # Cretes the menu
    sys.exit(app.exec_())  # Close the application

if __name__ == '__main__':
    main()
```

The figure below describes the different elements in the code. The purpose of the figure is to illustrate the iterations and relation amongst the objects.

Diagram 1. Classes and Methods for Section one



No.2 Creating a second option and print a message box upon request

We are going to use the previous code to add an extra option to the main menu. The purpose of this section is to show how to add a messagebox with the words “Hello World !!!” upon the selection of an option.

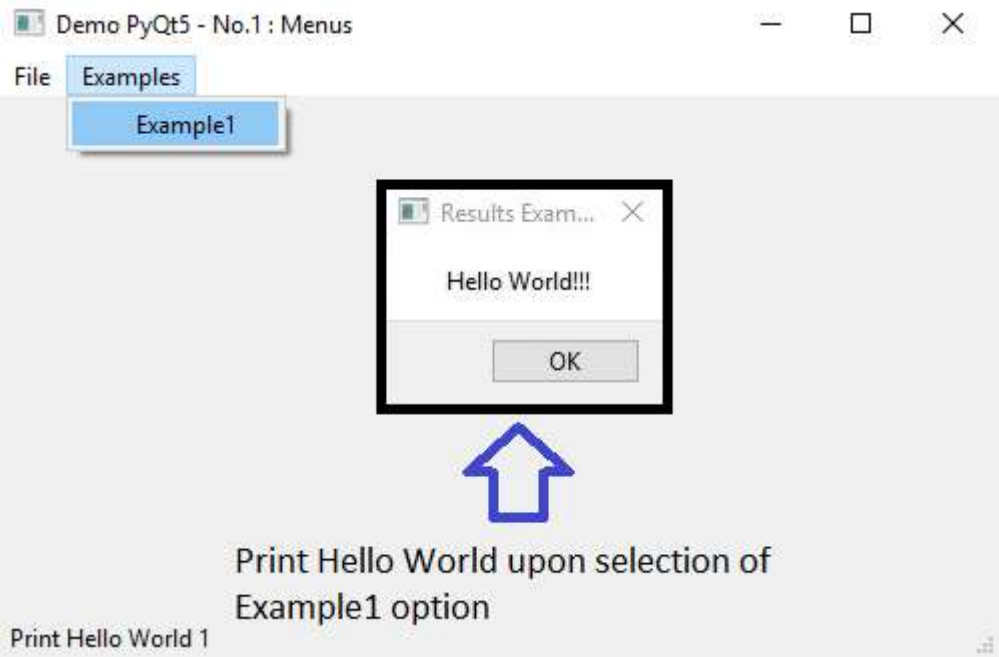


Figure 2. Hello World Message.

The code below shows the code to implement the new functionality. The additional code is highlighted in green. The only new command used in this section is QMessageBox. This new method is imported from the QtWidgets library by PyQt5.

Table 2. Code for Printing a "Hello World" Message

Download the code from:
https://github.com/saoderivera/PyQt5Tutorial/blob/master/TutorialPyQT502.py

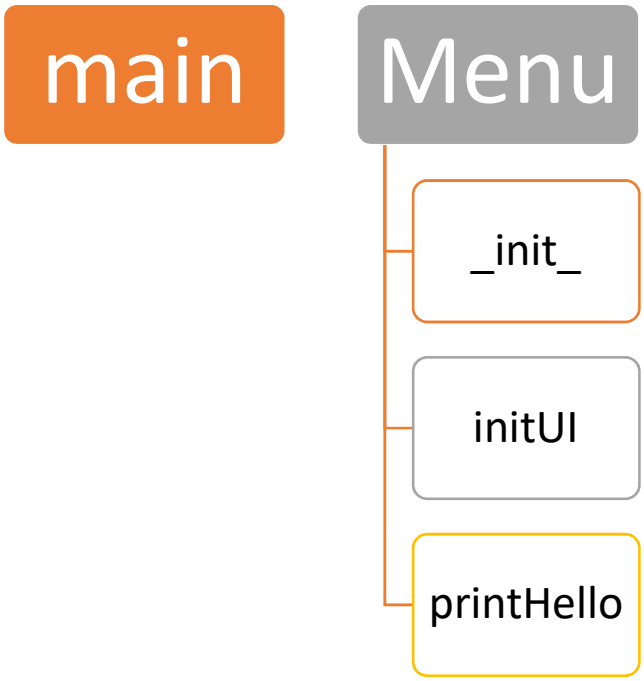
```
#::-------------------------------------------------------------------
#:: To create a Menu with options this are the libraries and components that
#:: requiered. For each new option we will be o adding new components
#::-------------------------------------------------------------------
import sys
from PyQt5.QtWidgets import QMainWindow, QAction, QMenu, QApplication
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QMessageBox

#::-------------------------------------------------------------------
#:: Definition of a Class for the main manu in the application
#::-------------------------------------------------------------------
class Menu(QMainWindow):

    def __init__(self):

        super().__init__()
        #::------------------------
        #:: variables use to set the size of the window that contains the menu
        #::------------------------
```

```
        self.left = 100
        self.top = 100
        self.width = 500
        self.height = 300

        #:: Title for the application

        self.Title = 'Demo PyQt5 - No.1 : Menus'

        #:: The initUi is call to create all the necessary elements for the menu

        self.initUI()

    def initUI(self):

        #::-------------------------------------------------
        # Creates the manu and the items
        #::-------------------------------------------------
        self.setWindowTitle(self.Title)
        self.setGeometry(self.left, self.top, self.width, self.height)
        self.statusBar()
        #::-----------------------------
        # 1. Create the menu bar
        # 2. Create an item in the menu bar
        # 3. Creaate an action to be executed the option in the  menu bar is choosen
        #::-----------------------------
        mainMenu = self.menuBar()
        fileMenu = mainMenu.addMenu('File')

        #:: Add another option to the Menu Bar

        exampleWin = mainMenu.addMenu ('Examples')

        #::-------------------------------------
        # Exit action
        # The following code creates the the da Exit Action along
        # with all the characteristics associated with the action
        # The Icon, a shortcut , the status tip that would appear in the window
        # and the action
        #  triggered.connect will indicate what is to be done when the item in
        # the menu is selected
        # These definitions are not available until the button is assigned
        # to the menu
        #::-------------------------------------

        exitButton = QAction(QIcon('enter.png'), '&Exit', self)
        exitButton.setShortcut('Ctrl+Q')
        exitButton.setStatusTip('Exit application')
        exitButton.triggered.connect(self.close)

        #:: This line adds the button (item element ) to the menu

        fileMenu.addAction(exitButton)

        #::-----------------------------------------------------
        #::Add Example 1 We create the item Menu Example1
        #::This option will present a message box upon request
        #::-----------------------------------------------------

        example1Button = QAction("Example1", self)
        example1Button.setStatusTip("Print Hello World 1")
        example1Button.triggered.connect(self.printhello)
```

```
        #:: We addd the example1Button action to the Menu Examples
        exampleWin.addAction(example1Button)

        #:: This line shows the windows

        self.show()

    def printhello(self):
        QMessageBox.about(self, "Results Example1", "Hello World!!!")

#::------------------------
#:: Application starts here
#::------------------------

def main():
    app = QApplication(sys.argv)  # creates the PyQt5 application
    mn = Menu()  # Cretes the menu
    sys.exit(app.exec_())  # Close the application

if __name__ == '__main__':
    main()
```

The figure below shows the new method and where it is located in the code.

Diagram 2. Classes and Methods for Section Two



No.3 Managing different types of Layouts

The purpose of this section is to show the mechanics of presenting information in three different layouts: vertical, horizontal and grid. We will be manipulating at least two windows at the same time, the menu windows and the window that presents information in the desired layout. PyQt uses “signals” to transfer the control from one window to another upon request. These windows create

communication with the user, to show data or ask for parameters to be used by the application, that is what is called “dialogs”. We will use a list of dialogs to keep track of the different iterations that are active in the application. Signals and the list of dialogs will allow us to manage different windows with different items at the same time. The figure below shows the application with the three types of layout that can be generated with PyQT.

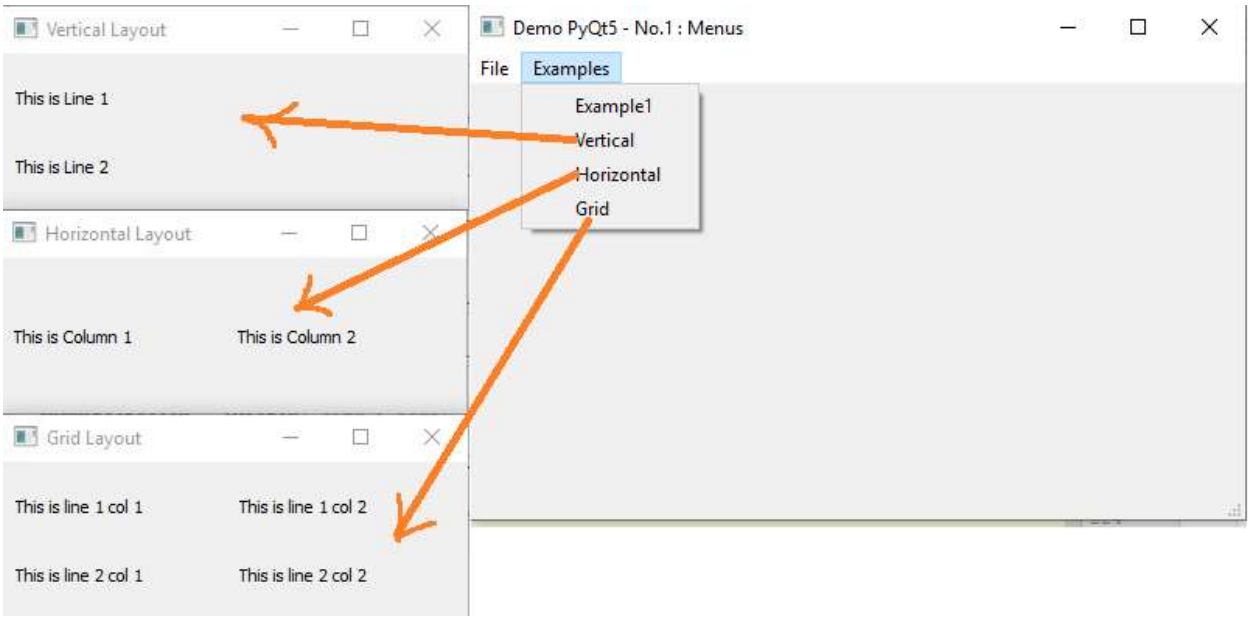


Figure 3. Layout Examples

To implement these layouts we introduce the use of

GridLoyout, QVBoxLayout, QHBoxLayout to manage the different presentations.

QLabel to print the different labels on the windows

pyqtSignal(str) to manage the signals amongst the different windows.

The following section presents the code that implements this new functionality. The new sections are in grey.

Table 3. Code for Different Layouts

Download the code from:

https://github.com/saoderivera/PyQt5Tutorial/blob/master/TutorialPyQt503.py

```
#::----------------------------------------------------------------
#:: To create a Menu with options this are the libraries and components that
#:: requiered. For each new option we will be o adding new components
#::----------------------------------------------------------------
import sys
from PyQt5.QtWidgets import QMainWindow, QAction, QMenu, QApplication
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QMessageBox    # No.2
```

```
from PyQt5.QtCore import pyqtSlot        # No. 3
from PyQt5.QtCore import pyqtSignal      # No. 3
from PyQt5.QtWidgets import  QWidget,QLabel, QVBoxLayout, QHBoxLayout, QGridLayout #
No. 3



#::-------------------------------------------------------------------------------
---
#:: Class Vertical Layout   # No. 3
#::-------------------------------------------------------------------------------
---
class VLayoutclass(QMainWindow):  ## All the class was added in No. 3 Section
    send_fig = pyqtSignal(str)  # To manage the signals PyQT manages the
communication

    def __init__(self):
        #::--------------------------------------------------------
        # Initialize the values of the class
        # Here the class inherits all the attributes and methods from the
QMainWindow
        #::--------------------------------------------------------
        super(VLayoutclass, self).__init__()

        self.Title = 'Vertical Layout'
        self.initUi()

    def initUi(self):
        #::--------------------------------------------------------------
        #  We create the type of layout QVBoxLayout (Vertical Layout )
        #  This type of layout comes from QWidget
        #::--------------------------------------------------------------
        self.setWindowTitle(self.Title)
        self.main_widget = QWidget(self)
        self.layout = QVBoxLayout(self.main_widget)    # Creates vertical layout
        self.label1 = QLabel("This is Line 1")         # Creates label1
        self.label2 = QLabel("This is Line 2")         # Creates label2
        self.layout.addWidget(self.label1)             # Add label 1 to  layout
        self.layout.addWidget(self.label2)             # Add label 2 to layout
        self.setCentralWidget(self.main_widget)        # Creates the window with all
the elements
        self.resize(300, 100)                          # Resize the window


#::-------------------------------------------------------------------------------
---
#:: Class Horizontal Layout   # No. 3
#::-------------------------------------------------------------------------------
---
class HLayoutclass(QMainWindow):  ## All the class was added in No. 3 Section
    send_fig = pyqtSignal(str)  # To manage the signals PyQT manages the
communication

    def __init__(self):
        #::--------------------------------------------------------
        # Initialize the values of the class
        # Here the class inherits all the attributes and methods from the
QMainWindow
        #::--------------------------------------------------------
        super(HLayoutclass, self).__init__()
```

```
        self.Title = 'Horizontal Layout'
        self.initUi()

    def initUi(self):
        #::-----------------------------------------------------------------
        #  We create the type of layout QHBoxLayout (Horizontal Layout )
        #  This type of layout comes from QWidget
        #::-----------------------------------------------------------------
        self.setWindowTitle(self.Title)
        self.main_widget = QWidget(self)
        self.layout = QHBoxLayout(self.main_widget)    # Creates horizontal layout
        self.label1 = QLabel("This is Column 1")         # Creates label1
        self.label2 = QLabel("This is Column 2")         # Creates label2
        self.layout.addWidget(self.label1)             # Add label 1 to  layout
        self.layout.addWidget(self.label2)             # Add label 2 to layout
        self.setCentralWidget(self.main_widget)        # Creates the window with all
the elements
        self.resize(300, 100)                          # Resize the window


#::-------------------------------------------------------------------------------
---
#:: Class Horizontal Layout   # No. 3
#::-------------------------------------------------------------------------------
---
class GLayoutclass(QMainWindow):  ## All the class was added in No. 3 Section
    send_fig = pyqtSignal(str)  # To manage the signals PyQT manages the
communication

    def __init__(self):
        #::-------------------------------------------------------
        # Initialize the values of the class
        # Here the class inherits all the attributes and methods from the
QMainWindow
        #::-------------------------------------------------------
        super(GLayoutclass, self).__init__()

        self.Title = 'Grid Layout'
        self.initUi()

    def initUi(self):
        #::-----------------------------------------------------------------
        #  We create the type of layout QGridLayout (Horizontal Layout )
        #  This type of layout comes from QWidget
        #::-----------------------------------------------------------------
        self.setWindowTitle(self.Title)
        self.main_widget = QWidget(self)
        self.layout = QGridLayout(self.main_widget)    # Creates horizontal layout
        self.label1 = QLabel("This is line 1 col 1")         # Creates label1
        self.label2 = QLabel("This is line 1 col 2")         # Creates label2
        self.label3 = QLabel("This is line 2 col 1")         # Creates label3
        self.label4 = QLabel("This is line 2 col 2")         # Creates label4
        self.layout.addWidget(self.label1,0,0)             # Add label 1 to  layout
        self.layout.addWidget(self.label2,0,1)             # Add label 2 to layout
        self.layout.addWidget(self.label3,1,0)             # Add label 3 to layout
        self.layout.addWidget(self.label4,1,1)             # Add label 4 to layout

        self.setCentralWidget(self.main_widget)        # Creates the window with all
the elements
        self.resize(300, 100)                          # Resize the window


#::-------------------------------------------------------------
```

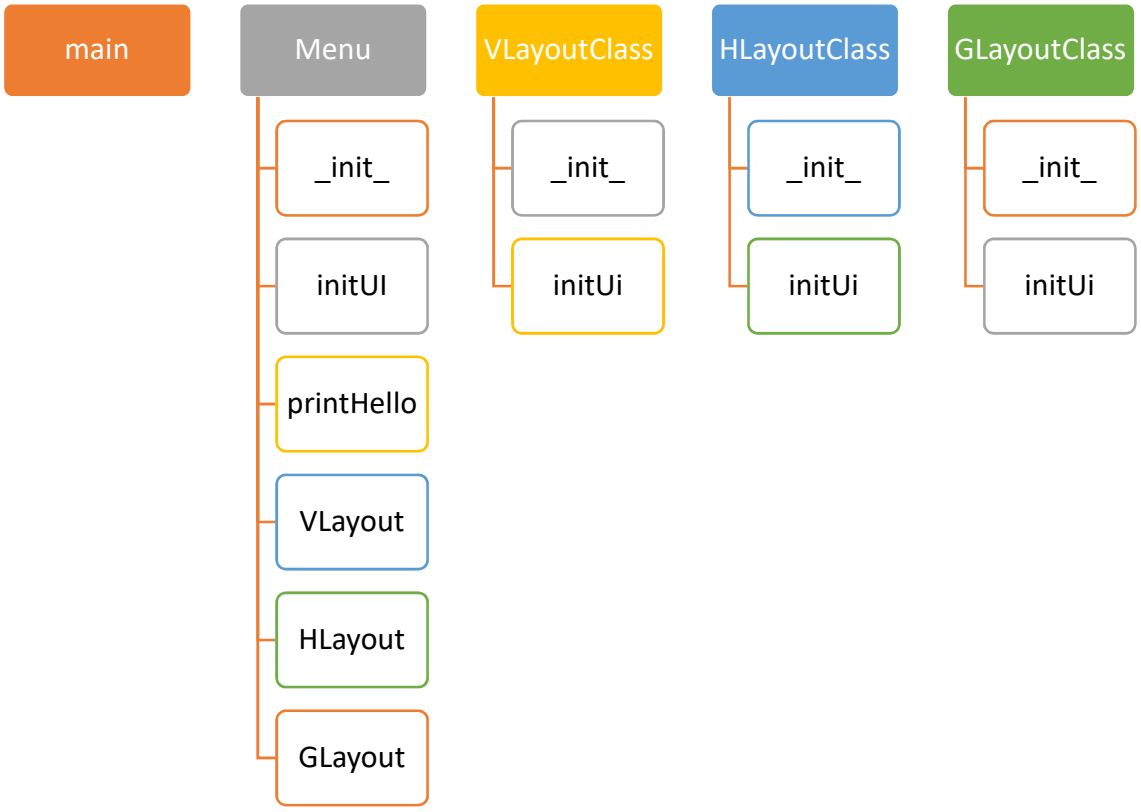
```
#:: Definition of a Class for the main manu in the application
#::------------------------------------------------------------
class Menu(QMainWindow):

    def __init__(self):

        super().__init__()
        #::----------------------
        #:: variables use to set the size of the window that contains the menu
        #::----------------------
        self.left = 100
        self.top = 100
        self.width = 500
        self.height = 300

        #:: Title for the application

        self.Title = 'Demo PyQt5 - No.1 : Menus'

        #:: The initUi is call to create all the necessary elements for the menu

        self.initUI()

    def initUI(self):

        #::-------------------------------------------------
        # Creates the manu and the items
        #::-------------------------------------------------
        self.setWindowTitle(self.Title)
        self.setGeometry(self.left, self.top, self.width, self.height)
        self.statusBar()
        #::-----------------------------
        # 1. Create the menu bar
        # 2. Create an item in the menu bar
        # 3. Creaate an action to be executed the option in the  menu bar is choosen
        #::-----------------------------
        mainMenu = self.menuBar()
        fileMenu = mainMenu.addMenu('File')

        #:: Add another option to the Menu Bar

        exampleWin = mainMenu.addMenu ('Examples')    # No. 2

        #::--------------------------------------
        # Exit action
        # The following code creates the the da Exit Action along
        # with all the characteristics associated with the action
        # The Icon, a shortcut , the status tip that would appear in the window
        # and the action
        #  triggered.connect will indicate what is to be done when the item in
        # the menu is selected
        # These definitions are not available until the button is assigned
        # to the menu
        #::--------------------------------------

        exitButton = QAction(QIcon('enter.png'), '&Exit', self)
        exitButton.setShortcut('Ctrl+Q')
        exitButton.setStatusTip('Exit application')
        exitButton.triggered.connect(self.close)

        #:: This line adds the button (item element ) to the menu

        fileMenu.addAction(exitButton)
```

```
        #::-----------------------------------------------------
        #::Add Example 1 We create the item Menu Example1
        #::This option will present a message box upon request
        #::-----------------------------------------------------

        example1Button = QAction("Example1", self)     # No. 2
        example1Button.setStatusTip("Print Hello World 1")    # No. 2
        example1Button.triggered.connect(self.printhello)     # No. 2

        #:: We addd the example1Button action to the Menu Examples
        exampleWin.addAction(example1Button)     # No. 2

        #::-------------------------------------------------------------
        #:: Add button for Vertical Layout    # No.3
        #::-------------------------------------------------------------

        example2Button = QAction("Vertical", self)     # No. 3
        example2Button.setStatusTip("Example of vertical layout")   # No. 3
        example2Button.triggered.connect(self.VLayout)   # No. 3

        #:: We addd the example2Button to the menu examples
        exampleWin.addAction(example2Button)     # No. 3

        #::-------------------------------------------------------------
        #:: Add button for Horizontal Layout    # No.3
        #::-------------------------------------------------------------

        example3Button = QAction("Horizontal", self)   # No. 3
        example3Button.setStatusTip("Example of horizontal layout")   # No. 3
        example3Button.triggered.connect(self.HLayout)   # No. 3

        #:: We addd the example2Button to the menu examples
        exampleWin.addAction(example3Button)   # No. 3

        #::-------------------------------------------------------------
        #:: Add button for Grid Layout    # No.3
        #::-------------------------------------------------------------

        example4Button = QAction("Grid", self)   # No. 3
        example4Button.setStatusTip("Example of Grid layout")   # No. 3
        example4Button.triggered.connect(self.GLayout)   # No. 3

        #:: We addd the example2Button to the menu examples
        exampleWin.addAction(example4Button)   # No. 3


        #:: Creates an empty list of dialogs to keep track of
        #:: all the iterations

        self.dialogs = list()

        #:: This line shows the windows
        self.show()

    def printhello(self):     # No. 2
        QMessageBox.about(self, "Results Example1", "Hello World!!!")   # No. 2

    def VLayout(self):   # No. 3
        dialog = VLayoutclass()   #  Creates an object with Vertical class
        self.dialogs.append(dialog)   # Appends the list of dialogs
        dialog.show()      # Show the window
```

```
    def HLayout(self):   # No. 3
        dialog = HLayoutclass()     # Creates an object with the Horizontal class
        self.dialogs.append(dialog) # Appeds the list of dialogs
        dialog.show()    # Show the window

    def GLayout(self):   # No. 3
        dialog = GLayoutclass()     # Creates an object with the Horizontal class
        self.dialogs.append(dialog) # Appeds the list of dialogs
        dialog.show()    # Show the window

#::------------------------
#:: Application starts here
#::------------------------

def main():
    app = QApplication(sys.argv)  # creates the PyQt5 application
    mn = Menu()  # Cretes the menu
    sys.exit(app.exec_())  # Close the application

if __name__ == '__main__':
    main()
```

The figure below shows the organization of the new methods.

Diagram 3. Classes and Methods for Section Three



No.4 Using controls in PyQT5: checkbox

The main objective of this section is showing how to use the control checkbox in a window, and how to to make code associate with action of “checked” and “un-checked”. The exercise presented here changes the title of the window. The checkbox is a widget that has two states: on and off. The widget is Qcheckbox that is imported from QWidgets. The figure below shows the output of the application. If the checkbox is un-checked it window will not the title: ”Title:Control Title.”

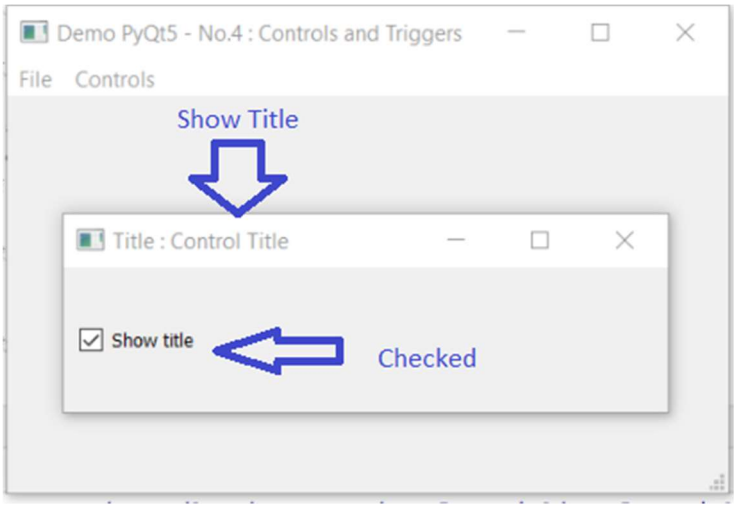


Figure 4. Checkbox Example

The QCheckBox from the library Widgets is used.

```
from PyQt5.QtWidgets import QCheckBox    # checkbox
```

To implement the checkbox we used the following code

```
Cbox = QCheckBox('Show title', self)  #the label on the right of the checkbox
Cbox.move(20, 20)                     # 20 pixels left , 20 pixels dawn the margins
Cbox.toggle()                         # contructor
Cbox.stateChanged.connect(self.changeTitle) # when the states changes from check to
un-checked and viceversa the method changeTitle is called.
```

The following is the entire code. The code associated with the widget is presented in gray. We do not go over the creation the menu option. Refer to the first section to check for the creation of menu options.

Table 4. Code for a Checkbox

Download the code from:
https://github.com/saoderivera/PyQt5Tutorial/blob/master/TutorialPyQt504.py

```
'''
In the application we will implement controls and triggers
the controls that oversee here are:
   Checkbox
```

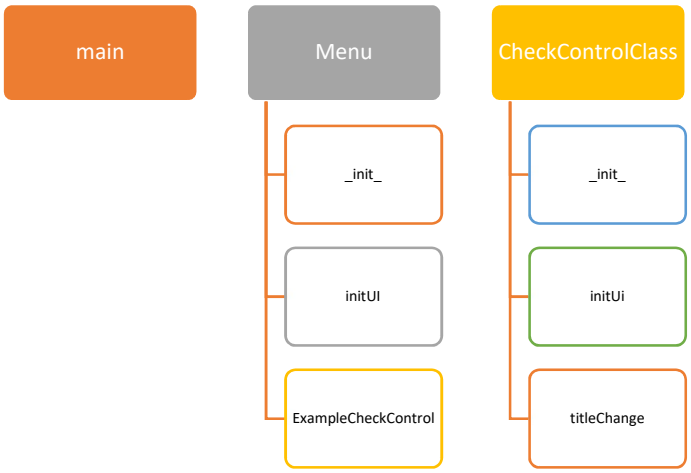
```
'''

import sys
from PyQt5.QtWidgets import QMainWindow, QAction, QMenu, QApplication
from PyQt5.QtWidgets import QCheckBox    # checkbox
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QMessageBox

from PyQt5.QtCore import pyqtSlot
from PyQt5.QtCore import pyqtSignal
from PyQt5.QtCore import Qt  # Control status
from PyQt5.QtWidgets import  QWidget,QLabel, QVBoxLayout, QHBoxLayout, QGridLayout

#::-----------------------------------------------------------------------------
---
#:: Class: Check Control
#::-----------------------------------------------------------------------------
---
class CheckControlClass(QMainWindow):
    send_fig = pyqtSignal(str)  # To manage the signals PyQT manages the
communication

    def __init__(self):
        #::-------------------------------------------------------------
        # Initialize the values of the class
        # Here the class inherits all the attributes and methods from the
QMainWindow
        #::-------------------------------------------------------------
        super(CheckControlClass, self).__init__()

        self.Title = 'Title : Control Title '
        self.initUi()

    def initUi(self):
        #::---------------------------------------------------------------------
        #  We create the type of layout QVBoxLayout (Vertical Layout )
        #  This type of layout comes from QWidget
        #::---------------------------------------------------------------------
        self.setWindowTitle(self.Title)
        self.main_widget = QWidget(self)
        self.layout = QVBoxLayout(self.main_widget)    # Creates vertical layout

        Cbox = QCheckBox('Show title', self)
        Cbox.move(20, 20)
        Cbox.toggle()
        Cbox.stateChanged.connect(self.changeTitle)

        self.setGeometry(300, 300, 250, 150)

        self.layout.addWidget(Cbox)                    # Add Check box to vertical
loyout
        self.setCentralWidget(self.main_widget)        # Creates the window with all
the elements
        self.resize(300, 100)                          # Resize the window


    def changeTitle(self, state):
        if state == Qt.Checked:
            self.setWindowTitle('Title : Control Title ')
        else:
            self.setWindowTitle(' ')
```

```
#::------------------------------------------------------------
#:: Definition of a Class for the main manu in the application
#::------------------------------------------------------------
class Menu(QMainWindow):

    def __init__(self):

        super().__init__()
        #::----------------------
        #:: variables use to set the size of the window that contains the menu
        #::----------------------
        self.left = 100
        self.top = 100
        self.width = 500
        self.height = 300

        #:: Title for the application

        self.Title = 'Demo PyQt5 - No.4 : Controls and Triggers'

        #:: The initUi is call to create all the necessary elements for the menu

        self.initUI()

    def initUI(self):

        #::-------------------------------------------------
        # Creates the manu and the items
        #::-------------------------------------------------
        self.setWindowTitle(self.Title)
        self.setGeometry(self.left, self.top, self.width, self.height)
        self.statusBar()
        #::-----------------------------
        # 1. Create the menu bar
        # 2. Create an item in the menu bar
        # 3. Creaate an action to be executed the option in the  menu bar is choosen
        #::-----------------------------
        mainMenu = self.menuBar()
        fileMenu = mainMenu.addMenu('File')

        #:: Add another option to the Menu Bar

        exampleWin = mainMenu.addMenu ('Controls')

        #::--------------------------------------
        # Exit action
        # The following code creates the the da Exit Action along
        # with all the characteristics associated with the action
        # The Icon, a shortcut , the status tip that would appear in the window
        # and the action
        #  triggered.connect will indicate what is to be done when the item in
        # the menu is selected
        # These definitions are not available until the button is assigned
        # to the menu
        #::--------------------------------------

        exitButton = QAction(QIcon('enter.png'), '&Exit', self)
        exitButton.setShortcut('Ctrl+Q')
        exitButton.setStatusTip('Exit application')
        exitButton.triggered.connect(self.close)

        #:: This line adds the button (item element ) to the menu
```

```
        fileMenu.addAction(exitButton)

        #::-------------------------------------------------------------
        #:: Add button to include a Checkbox
        #::-------------------------------------------------------------

        exampleCheckControl = QAction("Checkbox", self)
        exampleCheckControl.setStatusTip("Example of Checkbox")
        exampleCheckControl.triggered.connect(self.ExampleCheckControl)

        #:: We addd the exampleCheckControl to the menu examples
        exampleWin.addAction(exampleCheckControl)

        #:: Creates an empty list of dialogs to keep track of
        #:: all the iterations

        self.dialogs = list()

        #:: This line shows the windows
        self.show()


    def ExampleCheckControl(self):
        dialog = CheckControlClass()
        self.dialogs.append(dialog)  # Appends the list of dialogs
        dialog.show()      # Show the window

#::------------------------
#:: Application starts here
#::------------------------

def main():
    app = QApplication(sys.argv)  # creates the PyQt5 application
    mn = Menu()  # Cretes the menu
    sys.exit(app.exec_())  # Close the application

if __name__ == '__main__':
    main()
```

The figure below shows the organization of the methods used in this application.

Diagram 4. Classes and Methods for Section Four



No.5 Using controls in PyQT5: LineEdit and Pushbutton

The main objective of this section is showing how to use the control widget LineEdit in a window. This control allows us to enter information to be used later as label a in a graphic, or as parameters to create a graphic, for example. This application uses the ingested text to be copied into a label in the display window. The copy action will be implemented using a PushButton Widget.

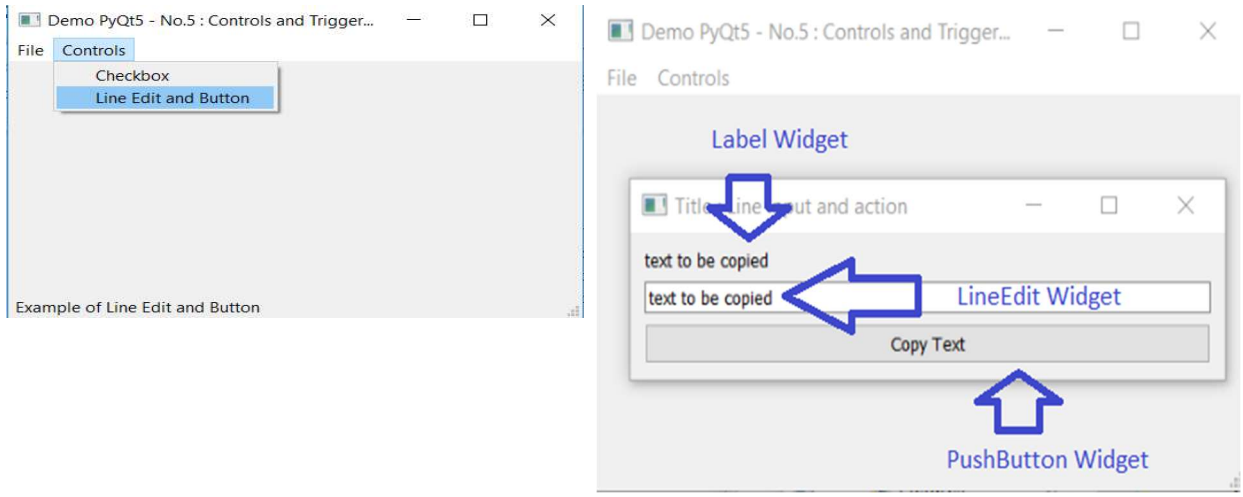


Figure 5. Line Edit and Button Example

The following widgets are used:

```
from PyQt5.QtWidgets import QPushButton   # pushbutton
from PyQt5.QtWidgets import QLineEdit     # Lineedit
from PyQt5.QtWidgets import QLevel        # label
```

The new option in the menu is added to the main menu

```
#::------------------------------------------------------------
#:: Add code to include Text Line and button to implement an action upon request
#::------------------------------------------------------------

exampleLEditButton =  QAction("Line Edit and Button", self)
exampleLEditButton.setStatusTip('Example of Line Edit and Button')
exampleLEditButton.triggered.connect(self.ExampleLEditButton)

exampleWin.addAction(exampleLEditButton)
```

The class “ExampleLeditButton” implements the window with the three widgets: label, pushbutton and lineedit. These items are added to the window and the method “CopyText” executes the copying. The windows that presents and ask for action uses a vertical layout.

```
self.exlabel = QLabel("<to be copied here>",self) # label
self.txtInputText = QLineEdit(self)        # LineEdit

self.btnCopyAction = QPushButton("Copy Text",self) # Pushbutton
self.btnCopyAction.clicked.connect(self.CopyText)  # On push call the copytext method
```

The method CopyText is call upon the action “clicked” from the button widget.

```
def CopyText(self):
    self.exlabel.setText(self.txtInputText.text())
```

This method only assigns the text to the label. The complete code is below. The code to implement this application is in gray color.

Table 5. Code for a LIneEdit and a PushButton

Download the code from:
https://github.com/saoderivera/PyQt5Tutorial/blob/master/TutorialPyQt505.py

```
'''
In the application we will implement controls and triggers
the controls that oversee here are:
   TextBox
'''

import sys
from PyQt5.QtWidgets import QMainWindow, QAction, QMenu, QApplication

from PyQt5.QtWidgets import QCheckBox    # checkbox
from PyQt5.QtWidgets import QPushButton  # pushbutton
from PyQt5.QtWidgets import QLineEdit    # Lineedit

from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QMessageBox

from PyQt5.QtCore import pyqtSlot
from PyQt5.QtCore import pyqtSignal
from PyQt5.QtCore import Qt  # Control status
from PyQt5.QtWidgets import  QWidget,QLabel, QVBoxLayout, QHBoxLayout, QGridLayout

#::-------------------------------------------------------------------------------
---
#:: Class: Check Control
#::-------------------------------------------------------------------------------
---
class CheckControlClass(QMainWindow):
    send_fig = pyqtSignal(str)  # To manage the signals PyQT manages the
communication

    def __init__(self):
        #::-------------------------------------------------------
        # Initialize the values of the class
        # Here the class inherits all the attributes and methods from the
QMainWindow
```

```
        #::-------------------------------------------------------
        super(CheckControlClass, self).__init__()

        self.Title = 'Title : Control Title '
        self.initUi()

    def initUi(self):
        #::-------------------------------------------------------------------
        #  We create the type of layout QVBoxLayout (Vertical Layout )
        #  This type of layout comes from QWidget
        #::-------------------------------------------------------------------
        self.setWindowTitle(self.Title)
        self.main_widget = QWidget(self)
        self.layout = QVBoxLayout(self.main_widget)    # Creates vertical layout

        Cbox = QCheckBox('Show title', self)
        Cbox.move(20, 20)
        Cbox.stateChanged.connect(self.ModifyTitle)

        self.setGeometry(300, 300, 250, 150)

        self.layout.addWidget(Cbox)

        self.setCentralWidget(self.main_widget)       # Creates the window with all
the elements
        self.resize(300, 100)                         # Resize the window


    def ModifyTitle(self, state):
        if state == Qt.Checked:
            self.setWindowTitle('Title : Control Title ')
        else:
            self.setWindowTitle(' ')



#::-------------------------------------------------------------------------------------
---
#:: Class: Line Edit Control
#::-------------------------------------------------------------------------------------
---
class LEditButtonClass(QMainWindow):
    send_fig = pyqtSignal(str)  # To manage the signals PyQT manages the
communication

    def __init__(self):
        #::-------------------------------------------------------
        # Initialize the values of the class
        # Here the class inherits all the attributes and methods from the
QMainWindow
        #::-------------------------------------------------------
        super(LEditButtonClass, self).__init__()

        self.Title = 'Title : Line input and action '
        self.initUi()

    def initUi(self):
        #::-------------------------------------------------------------------
        #  We create the type of layout QVBoxLayout (Vertical Layout )
        #  This type of layout comes from QWidget
        #::-------------------------------------------------------------------
        self.setWindowTitle(self.Title)
        self.main_widget = QWidget(self)
```

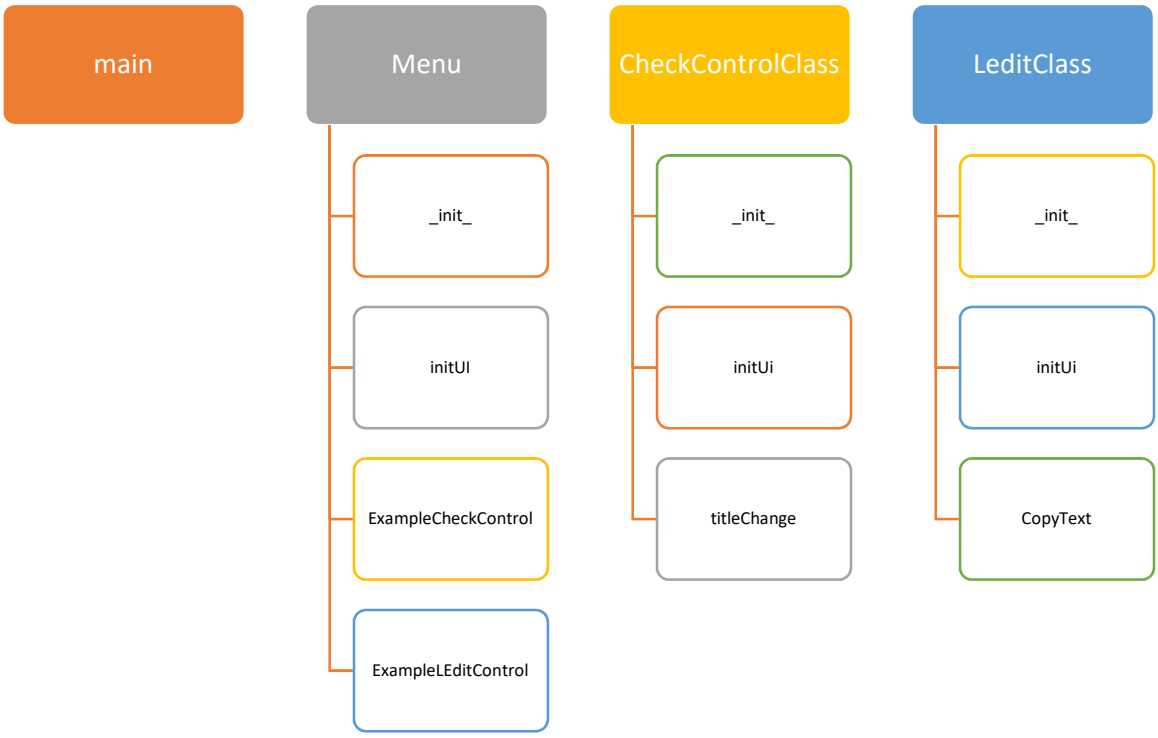
```
        self.layout = QVBoxLayout(self.main_widget)    # Creates vertical layout

        self.exlabel = QLabel("<to be copied here>",self) # exlabel can be use in
all the methods in the this class
        self.txtInputText = QLineEdit(self)

        self.btnCopyAction = QPushButton("Copy Text",self)
        self.btnCopyAction.clicked.connect(self.CopyText)

        self.layout.addWidget(self.exlabel)
        self.layout.addWidget(self.txtInputText)
        self.layout.addWidget(self.btnCopyAction)
        self.setGeometry(300, 300, 250, 150)

        self.setCentralWidget(self.main_widget)       # Creates the window with all
the elements
        self.resize(300, 100)                         # Resize the window


    def CopyText(self):
        self.exlabel.setText(self.txtInputText.text())

#::-------------------------------------------------------------
#:: Definition of a Class for the main manu in the application
#::-------------------------------------------------------------
class Menu(QMainWindow):

    def __init__(self):

        super().__init__()
        #::------------------------
        #:: variables use to set the size of the window that contains the menu
        #::------------------------
        self.left = 100
        self.top = 100
        self.width = 500
        self.height = 300

        #:: Title for the application

        self.Title = 'Demo PyQt5 - No.5 : Controls and Triggers II'

        #:: The initUi is call to create all the necessary elements for the menu

        self.initUI()

    def initUI(self):

        #::-----------------------------------------------
        # Creates the manu and the items
        #::-----------------------------------------------
        self.setWindowTitle(self.Title)
        self.setGeometry(self.left, self.top, self.width, self.height)
        self.statusBar()
        #::-----------------------------
        # 1. Create the menu bar
        # 2. Create an item in the menu bar
        # 3. Creaate an action to be executed the option in the  menu bar is choosen
        #::-----------------------------
        mainMenu = self.menuBar()
        fileMenu = mainMenu.addMenu('File')

        #:: Add another option to the Menu Bar
```

```
        exampleWin = mainMenu.addMenu ('Controls')

        #::--------------------------------------
        # Exit action
        # The following code creates the the da Exit Action along
        # with all the characteristics associated with the action
        # The Icon, a shortcut , the status tip that would appear in the window
        # and the action
        #  triggered.connect will indicate what is to be done when the item in
        # the menu is selected
        # These definitions are not available until the button is assigned
        # to the menu
        #::--------------------------------------

        exitButton = QAction(QIcon('enter.png'), '&Exit', self)
        exitButton.setShortcut('Ctrl+Q')
        exitButton.setStatusTip('Exit application')
        exitButton.triggered.connect(self.close)

        #:: This line adds the button (item element ) to the menu

        fileMenu.addAction(exitButton)

        #::-------------------------------------------------------------
        #:: Add code to include a Checkbox
        #::-------------------------------------------------------------

        exampleCheckControl = QAction("Checkbox", self)
        exampleCheckControl.setStatusTip("Example of Checkbox")
        exampleCheckControl.triggered.connect(self.ExampleCheckControl)

        #:: We add the exampleCheckControl to the menu examples
        exampleWin.addAction(exampleCheckControl)

        #::-------------------------------------------------------------
        #:: Add code to include Text Line and button to implement an action upon
request
        #::-------------------------------------------------------------

        exampleLEditButton =  QAction("Line Edit and Button", self)
        exampleLEditButton.setStatusTip('Example of Line Edit and Button')
        exampleLEditButton.triggered.connect(self.ExampleLEditButton)

        exampleWin.addAction(exampleLEditButton)


        #:: Creates an empty list of dialogs to keep track of
        #:: all the iterations

        self.dialogs = list()

        #:: This line shows the windows
        self.show()


    def ExampleCheckControl(self):
        dialog = CheckControlClass()
        self.dialogs.append(dialog)  # Appends to the list of dialogs
        dialog.show()      # Show the window

    def ExampleLEditButton(self):
        dialog = LEditButtonClass()
```

```
        self.dialogs.append(dialog) # Apppends to the list of dialogs
        dialog.show()

#::------------------------
#:: Application starts here
#::------------------------

def main():
    app = QApplication(sys.argv)  # creates the PyQt5 application
    mn = Menu()  # Cretes the menu
    sys.exit(app.exec_())  # Close the application

if __name__ == '__main__':
    main()
```

The figure below shows the organization of the methods used in this application.

Diagram 5. Classes and Methods for Section Five



No.6 Using controls in PyQT5: Radio Buttons

The purpose of this section is showing how to use radio buttons. Radio buttons are very useful to choose parameters. First you create the radio buttons and PyQt5 will keep track of which one is selected. This

application will show how to create three radio buttons and how to use selected parameter to print a message on label widget. The following figures shows the application.

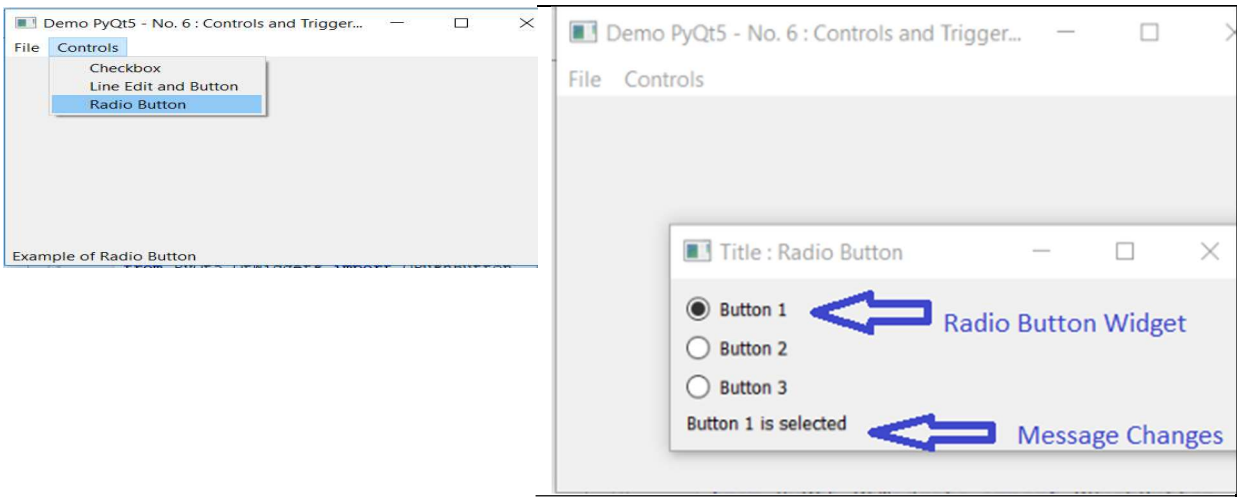


Figure 6. Radio Button Example

The message label will change upon selection of the radio button: Button1 is selected, Button2 is selected, and Button3 is selected. All the radio buttons created in the window interact with each other which means that only one button can be selected at a time. If there is the need of having two sets of buttons or more, these buttons should be arranged in groups. The groupbox widget will be review in the next section.

The new widget used is QRadioButton

```
from PyQt5.QtWidgets import QRadioButton # Radio Buttons
```

The application will create three buttons, which will trigger an action upon selection. This action is the calling to the method “onClicked”. This method in turn will change the message displayed in the label presented at the bottom of the window. The buttons are arranged in a vertical layout.

```
self.b1 = QRadioButton("Button 1")          #Creation of first button
self.b1.setChecked(True)                    #This button is selected by default.
self.b1.toggled.connect(self.onClicked)     #call upon selection

self.b2 = QRadioButton("Button 2")          #creation of second button
self.b2.toggled.connect(self.onClicked)     #call upon selection

self.b3 = QRadioButton("Button 3")          #creation of third button
self.b3.toggled.connect(self.onClicked)     #call upon selection

self.buttonlabel= QLabel('Button 1 is selected',self) #Because the fist button is
select by default the message refers to the fist button selected.
```

The following table shows the code for the implementation of this application. The new code is colored in gray.

Table 6. Code for RadioButtons

Download code from: https://github.com/saoderivera/PyQt5Tutorial/blob/master/TutorialPyQt506.py

```
'''
In the application we will implement controls and triggers
the controls that oversee here are:
   Checkbox
   TextBox
   Radio buttoms
'''

import sys
from PyQt5.QtWidgets import QMainWindow, QAction, QMenu, QApplication

from PyQt5.QtWidgets import QCheckBox    # checkbox
from PyQt5.QtWidgets import QPushButton  # pushbutton
from PyQt5.QtWidgets import QLineEdit    # Lineedit
from PyQt5.QtWidgets import QRadioButton # Radio Buttons

from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QMessageBox

from PyQt5.QtCore import pyqtSlot
from PyQt5.QtCore import pyqtSignal
from PyQt5.QtCore import Qt  # Control status
from PyQt5.QtWidgets import  QWidget,QLabel, QVBoxLayout, QHBoxLayout, QGridLayout

#::-------------------------------------------------------------------------------
---
#:: Class: Check Control
#::-------------------------------------------------------------------------------
---
class CheckControlClass(QMainWindow):
    send_fig = pyqtSignal(str)  # To manage the signals PyQT manages the
communication

    def __init__(self):
        #::-----------------------------------------------------------
        # Initialize the values of the class
        # Here the class inherits all the attributes and methods from the
QMainWindow
        #::-----------------------------------------------------------
        super(CheckControlClass, self).__init__()

        self.Title = 'Title : Control Title '
        self.initUi()

    def initUi(self):
        #::-----------------------------------------------------------------
        #  We create the type of layout QVBoxLayout (Vertical Layout )
        #  This type of layout comes from QWidget
        #::-----------------------------------------------------------------
        self.setWindowTitle(self.Title)
        self.main_widget = QWidget(self)
        self.layout = QVBoxLayout(self.main_widget)    # Creates vertical layout

        Cbox = QCheckBox('Show title', self)
        Cbox.move(20, 20)
```

```
        Cbox.stateChanged.connect(self.ModifyTitle)

        self.setGeometry(300, 300, 250, 150)

        self.layout.addWidget(Cbox)

        self.setCentralWidget(self.main_widget)       # Creates the window with all
the elements
        self.resize(300, 100)                         # Resize the window


    def ModifyTitle(self, state):
        if state == Qt.Checked:
            self.setWindowTitle('Title : Control Title ')
        else:
            self.setWindowTitle(' ')


#::-------------------------------------------------------------------------------
---
#:: Class: Line Edit Control
#::-------------------------------------------------------------------------------
---
class LEditButtonClass(QMainWindow):
    send_fig = pyqtSignal(str)  # To manage the signals PyQT manages the
communication

    def __init__(self):
        #::-------------------------------------------------------
        # Initialize the values of the class
        # Here the class inherits all the attributes and methods from the
QMainWindow
        #::-------------------------------------------------------
        super(LEditButtonClass, self).__init__()

        self.Title = 'Title : Line input and action '
        self.initUi()

    def initUi(self):
        #::-------------------------------------------------------------
        #  We create the type of layout QVBoxLayout (Vertical Layout )
        #  This type of layout comes from QWidget
        #::-------------------------------------------------------------
        self.setWindowTitle(self.Title)
        self.main_widget = QWidget(self)
        self.layout = QVBoxLayout(self.main_widget)    # Creates vertical layout

        self.exlabel = QLabel("<to be copied here>",self) # exlabel can be use in
all the methods in the this class
        self.txtInputText = QLineEdit(self)

        self.btnCopyAction = QPushButton("Copy Text",self)
        self.btnCopyAction.clicked.connect(self.CopyText)

        self.layout.addWidget(self.exlabel)
        self.layout.addWidget(self.txtInputText)
        self.layout.addWidget(self.btnCopyAction)
        self.setGeometry(300, 300, 250, 150)

        self.setCentralWidget(self.main_widget)       # Creates the window with all
the elements
        self.resize(300, 100)                         # Resize the window
```

```
    def ChangeTitle(self, state):
        if state == Qt.Checked:
            self.setWindowTitle('Title : Control Title ')
        else:
            self.setWindowTitle(' ')


    def CopyText(self):
        self.exlabel.setText(self.txtInputText.text())

#::-------------------------------------------------------------------------------------
---
#:: Class: Radio Button
#::-------------------------------------------------------------------------------------
---
class RadioButtonClass(QMainWindow):
    send_fig = pyqtSignal(str)  # To manage the signals PyQT manages the
communication

    def __init__(self):
        #::-------------------------------------------------------------
        # Initialize the values of the class
        # Here the class inherits all the attributes and methods from the
QMainWindow
        #::-------------------------------------------------------------
        super(RadioButtonClass, self).__init__()

        self.Title = 'Title : Radio Button '
        self.initUi()

    def initUi(self):
        #::-----------------------------------------------------------------
        #  We create the type of layout QVBoxLayout (Vertical Layout )
        #  This type of layout comes from QWidget
        #::-----------------------------------------------------------------
        self.setWindowTitle(self.Title)
        self.main_widget = QWidget(self)
        self.layout = QVBoxLayout(self.main_widget)    # Creates vertical layout

        self.b1 = QRadioButton("Button 1")
        self.b1.setChecked(True)
        self.b1.toggled.connect(self.onClicked)
        self.layout.addWidget(self.b1)

        self.b2 = QRadioButton("Button 2")
        self.b2.toggled.connect(self.onClicked)
        self.layout.addWidget(self.b2)

        self.b3 = QRadioButton("Button 3")
        self.b3.toggled.connect(self.onClicked)
        self.layout.addWidget(self.b3)

        self.buttonlabel= QLabel('Button 1 is selected',self)
        self.layout.addWidget(self.buttonlabel)

        self.setCentralWidget(self.main_widget)        # Creates the window with all
the elements
        self.resize(300, 100)                          # Resize the window


    def onClicked(self):
        button = self.sender()
```

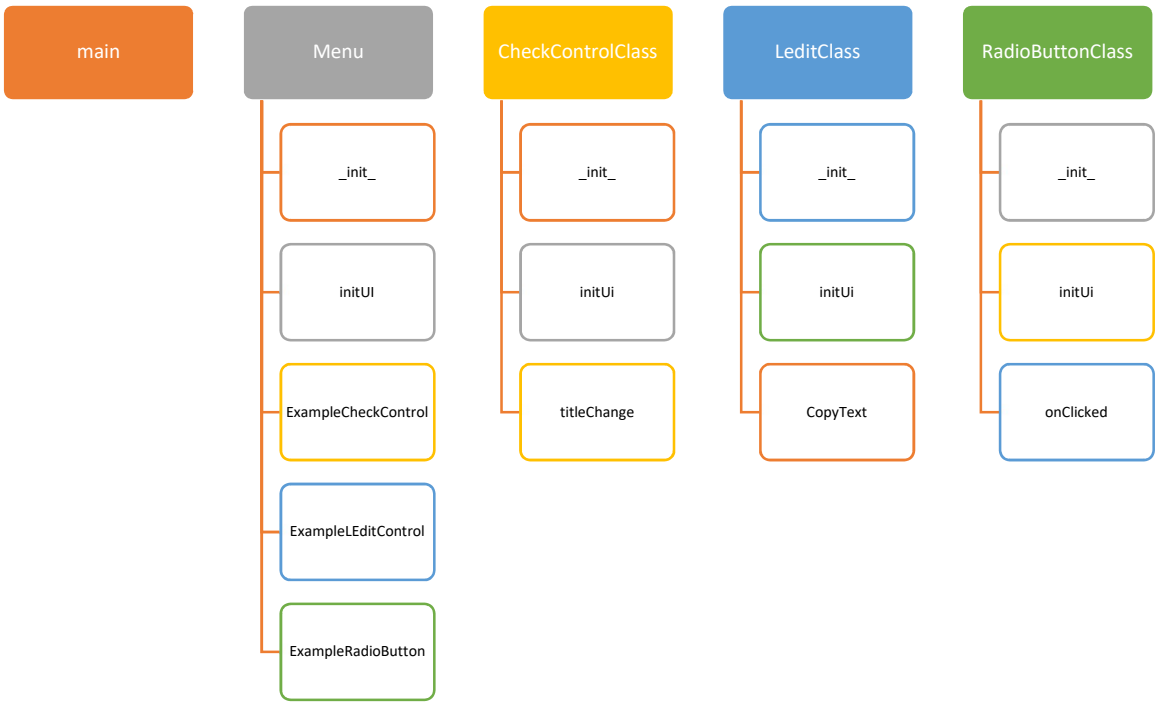
```
        if button.isChecked():
            self.buttonlabel.setText(button.text()+' is selected')




#::-------------------------------------------------------------
#:: Definition of a Class for the main manu in the application
#::-------------------------------------------------------------
class Menu(QMainWindow):

    def __init__(self):

        super().__init__()
        #::-----------------------
        #:: variables use to set the size of the window that contains the menu
        #::-----------------------
        self.left = 100
        self.top = 100
        self.width = 500
        self.height = 300

        #:: Title for the application

        self.Title = 'Demo PyQt5 - No. 6 : Controls and Triggers II'

        #:: The initUi is call to create all the necessary elements for the menu

        self.initUI()

    def initUI(self):

        #::-----------------------------------------------
        # Creates the manu and the items
        #::-----------------------------------------------
        self.setWindowTitle(self.Title)
        self.setGeometry(self.left, self.top, self.width, self.height)
        self.statusBar()
        #::----------------------------
        # 1. Create the menu bar
        # 2. Create an item in the menu bar
        # 3. Creaate an action to be executed the option in the  menu bar is choosen
        #::----------------------------
        mainMenu = self.menuBar()
        fileMenu = mainMenu.addMenu('File')

        #:: Add another option to the Menu Bar

        exampleWin = mainMenu.addMenu ('Controls')

        #::-------------------------------------
        # Exit action
        # The following code creates the the da Exit Action along
        # with all the characteristics associated with the action
        # The Icon, a shortcut , the status tip that would appear in the window
        # and the action
        #  triggered.connect will indicate what is to be done when the item in
        # the menu is selected
        # These definitions are not available until the button is assigned
        # to the menu
        #::-------------------------------------
```

```
        exitButton = QAction(QIcon('enter.png'), '&Exit', self)
        exitButton.setShortcut('Ctrl+Q')
        exitButton.setStatusTip('Exit application')
        exitButton.triggered.connect(self.close)

        #:: This line adds the button (item element ) to the menu

        fileMenu.addAction(exitButton)

        #::-------------------------------------------------------------
        #:: Add code to include a Checkbox
        #::-------------------------------------------------------------

        exampleCheckControl = QAction("Checkbox", self)
        exampleCheckControl.setStatusTip("Example of Checkbox")
        exampleCheckControl.triggered.connect(self.ExampleCheckControl)

        #:: We add the exampleCheckControl to the menu examples
        exampleWin.addAction(exampleCheckControl)

        #::-------------------------------------------------------------
        #:: Add code to include Text Line and button to implement an action upon
request
        #::-------------------------------------------------------------

        exampleLEditButton =  QAction("Line Edit and Button", self)
        exampleLEditButton.setStatusTip('Example of Line Edit and Button')
        exampleLEditButton.triggered.connect(self.ExampleLEditButton)

        exampleWin.addAction(exampleLEditButton)

        #::-------------------------------------------------------------
        #:: Add code to include radio buttons  to implement an action upon request
        #::-------------------------------------------------------------

        exampleRadioButton =  QAction("Radio Button ", self)
        exampleRadioButton.setStatusTip('Example of Radio Button')
        exampleRadioButton.triggered.connect(self.ExampleRadioButton)

        exampleWin.addAction(exampleRadioButton)

        #:: Creates an empty list of dialogs to keep track of
        #:: all the iterations

        self.dialogs = list()

        #:: This line shows the windows
        self.show()


    def ExampleCheckControl(self):
        dialog = CheckControlClass()
        self.dialogs.append(dialog)  # Appends to the list of dialogs
        dialog.show()      # Show the window

    def ExampleLEditButton(self):
        dialog = LEditButtonClass()
        self.dialogs.append(dialog) # Apppends to the list of dialogs
        dialog.show()

    def ExampleRadioButton(self):
        dialog = RadioButtonClass()
        self.dialogs.append(dialog) # Apppends to the list of dialogs
```

```
        dialog.show()

#::------------------------
#:: Application starts here
#::------------------------

def main():
    app = QApplication(sys.argv)  # creates the PyQt5 application
    mn = Menu()  # Cretes the menu
    sys.exit(app.exec_())  # Close the application

if __name__ == '__main__':
    main()
```

The diagram below shows the organization of the methods used in this application.

Diagram 6. Classes and Methods for Section Six



No.7 Putting everything together: GroupBox, MatplotLib , and a Graphic with parameters.

In this section groupbox widget will be described. This widget is important to group elements that are alike of have a conceptual meaning. It is also important when there is the need to have two or more sets of radio buttons in the same application, if they are not grouped together with the groupbox widget they will behave as one set, and only one button can be checked. Grouping information into meaningful concepts will help the users to understand what is being asked from them.

The groupbox widget is a container, thus a layout should be assigned. Once the layout has been chosen—horizontal, vertical, or grid, others widgets can be assingned and they will be displayed according to the requested layout.

The new widget introduced in this sections are

```
from PyQt5.QtWidgets import QGroupBox     # Group Box


# These components are essential for creating the graphics in pqt5
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as
NavigationToolbar
from matplotlib.figure import Figure      # Figure
```

Figure is used to create matplotlib graphs and FigureCanvas is used to position the graph on a window in the screen. The creation of the following examples will follow this process:

a) Create a window widget
b) Add layout to the window (Horizontal, Vertical, Grid)
c) Add group Box (one or more)
d) Add layout to the groupbox widget (Horizontal, Vertical, Grid)
e) Add widgets to the different groupboxes.

It is important that the user interface be planned in advance, since every component has to be put in the right place. A good approach is to make a mock-up of the application before starting to draw the components on the screen.

This application has three components: a windows widget with a groupbox; another window with a matplot graphic depicting a scatterplot; and lastly a window with graphic with parameters.

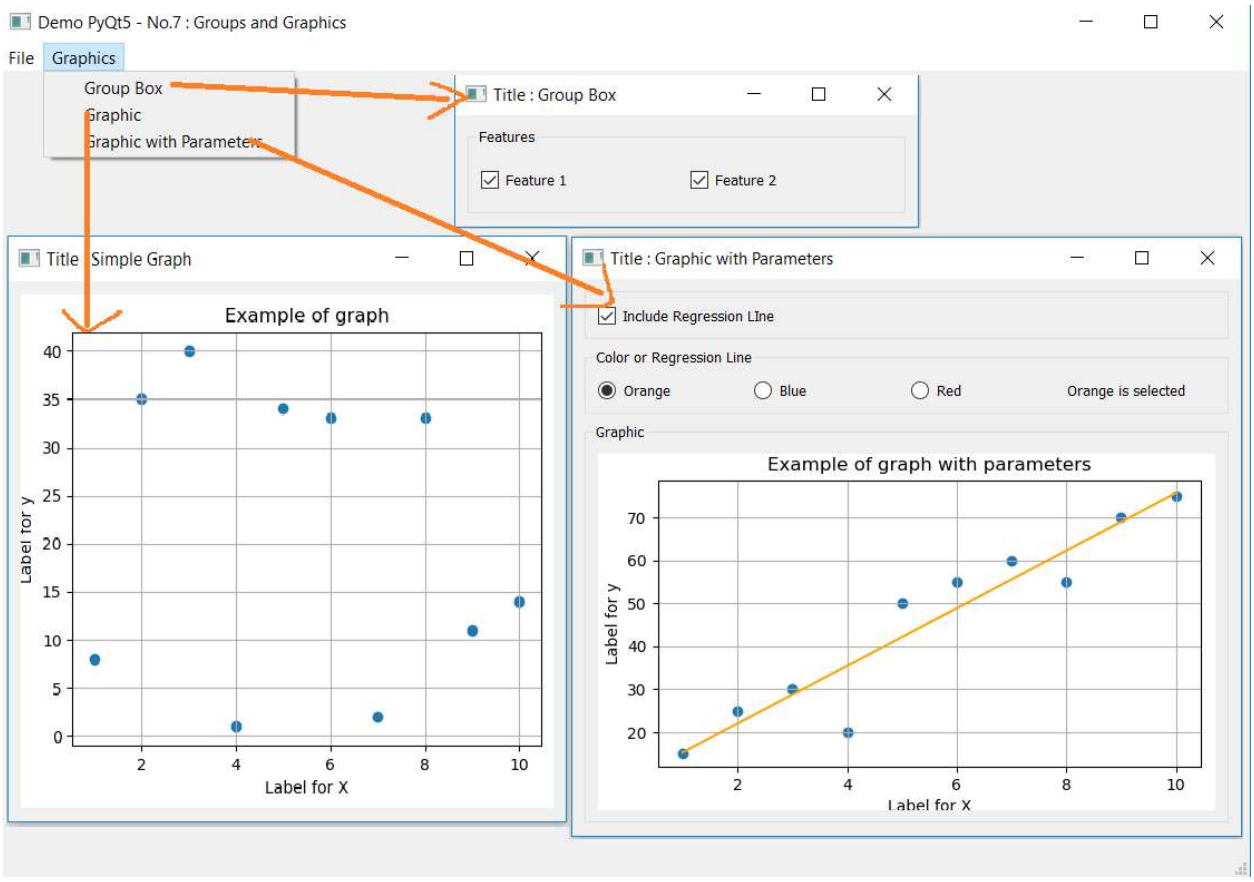


Figure 7. GroupBox and Graphics

The first option will generate a groupbox with horizontal layout displaying two cheboxes. The second option in the menu will present a graph with no parameters, the graph aims to present the use of figure and FigureCanvas. The last option implements thre groupboxes. The fist group just ask for the presentation of a regression line, the second group ask for the color of the regression line, and the last group contains the figure and canvasfigure for the graph using the chosen parameters.

First Option: GroupBox

Here is the important code for the groupbox implementation

```
self.groupBox1 = QGroupBox('Features')  # Creation of the group and the title
self.groupBox1Layout= QGridLayout()     # creation of the layout
self.groupBox1.setLayout(self.groupBox1Layout) #Assign the layout to the groupbox


self.Cbox1 = QCheckBox('Feature 1', self) #creation of the checkbox1
self.Cbox1.setChecked(True)               # checked by default
self.Cbox2 = QCheckBox('Feature 2', self)  #creation of the checkbox2
self.Cbox2.setChecked(True)               # checked by default
self.groupBox1Layout.addWidget(self.Cbox1, 0, 0)   #add checkbox 1 to the groupbox
self.groupBox1Layout.addWidget(self.Cbox2, 0, 1)   #add checkbox 2 to the groupbox

self.layout.addWidget(self.groupBox1)      # add the groupbox to the layout of the
window widget.
```

The fist option do not executes an action, it aims to show how to organize widgets into groups.

Second Option: Graphic

Here is the important code for the graphic implementation.

```
self.fig = Figure()               # Creates the object that contains the graphic
self.ax1 = self.fig.add_subplot(111)#adds a subplot the figure
self.canvas = FigureCanvas(self.fig)#create the canvas where the picture is presented

self.canvas.setSizePolicy(QSizePolicy.Expanding,
                          QSizePolicy.Expanding) #sizen parameters of the canvas

self.canvas.updateGeometry()      #refresh the sizing parametres

X_1 = [1,2,3,4,5,6,7,8,9,10]      #dummy information for the plot
y_1 = [8,35,40,1,34,33,2,33,11,14]

self.ax1.scatter(X_1,y_1)         #draws the information using the scatter plot

vtitle = "Example of graph"       #title of the graphic
self.ax1.set_title(vtitle)
self.ax1.set_xlabel("Label for X")         #Labelx
self.ax1.set_ylabel("Label for y")         #labely
self.ax1.grid(True)

self.fig.tight_layout()                    #adjust margins to the figure size
self.fig.canvas.draw_idle()                #preeents the scatter plot on the canvas


self.layout.addWidget(self.canvas)         #add the canvas with the plot to the layout
of the window, ready to view for the user.
```

Third Option: Graphic with Parameters.

Here is the important code for the graphic with parameters implementation.

First the groupboxes are created.

```
self.groupBox1 = QGroupBox('')
self.groupBox1Layout= QVBoxLayout()
self.groupBox1.setLayout(self.groupBox1Layout)

self.groupBox2 = QGroupBox('Color or Regression Line')
self.groupBox2Layout = QHBoxLayout()
self.groupBox2.setLayout(self.groupBox2Layout)

self.groupBox3 = QGroupBox('Graphic')
self.groupBox3Layout = QVBoxLayout()
self.groupBox3.setLayout(self.groupBox3Layout)
```

Then the components for each group are created: the component for the first group, a checkbox.

```
self.chkline = QCheckBox("Include Regression LIne", self) # Creation of checkbox
self.chkline.setChecked(True)                               # by default is checked
self.chkline.stateChanged.connect(self.onClicked)  # Method onClicked  when states
changes

self.groupBox1Layout.addWidget(self.chkline) # The widget is added to the group1
```

Then the components for the second group: Three radio buttons and a label

```
self.b1 = QRadioButton("Orange")           # Button 1
self.b1.setChecked(True)                   # checked by default
self.b1.toggled.connect(self.onClicked)    # calls onClicked when states changes

self.b2 = QRadioButton("Blue")
self.b2.toggled.connect(self.onClicked)    # calls onClicked when states changes


self.b3 = QRadioButton("Red")
self.b3.toggled.connect(self.onClicked)    # calls onClicked when states changes

self.buttonlabel = QLabel(self.vcolor+' is selected')  #label is created

self.groupBox2Layout.addWidget(self.b1)    # Add buttons to the secondBox
self.groupBox2Layout.addWidget(self.b2)
self.groupBox2Layout.addWidget(self.b3)
self.groupBox2Layout.addWidget(self.buttonlabel) # Add label secondBox
```

Lastly the third group of widgets are created and added to the corresponding layout.

```
# figure and canvas figure to draw the graph is created to

self.fig = Figure()                        # creates figure
self.ax1 = self.fig.add_subplot(111)       # subplot
self.canvas = FigureCanvas(self.fig)       # creates FigureCanvas that contains the
plot

self.canvas.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

self.canvas.updateGeometry()

# Canvas is added to the third group box
self.groupBox3Layout.addWidget(self.canvas)        # Add canvas to third group
```

The method onClicked is called when the radio buttons or the checkbox states changes. This method draws the plot using the selected parameters, the regression line, and the color of the regression line.

```
def onClicked(self):

    # Figure is cleared to create the new graph with the choosen parameters
    self.ax1.clear()

    # the buttons are inspect to indicate which one is checked.
    # vcolor is assigned the chosen color
    if self.b1.isChecked():
        self.vcolor = self.b1.text()
    if self.b2.isChecked():
        self.vcolor = self.b2.text()
    if self.b3.isChecked():
        self.vcolor = self.b3.text()

    # the label that displays the selected option
    self.buttonlabel.setText(self.vcolor+' is selected')
```

```
    # create the scatter plot , a radio button option could be created
    # to choose a scatter plot or other type of graphic

    self.ax1.scatter(self.X_1, self.y_1)

    # if checkbox for showing regression line is checked
    if self.chkline.isChecked():
        self.ax1.plot(self.X_1, self.b + self.m * self.X_1, '-', color=self.vcolor)

    vtitle = "Example of graph with parameters"
    self.ax1.set_title(vtitle)
    self.ax1.set_xlabel("Label for X")
    self.ax1.set_ylabel("Label for y")
    self.ax1.grid(True)

    # show the plot
    self.fig.tight_layout()
    self.fig.canvas.draw_idle()
```

This is how a plot with parameters can be implemented. It would be a good exercise to make a radio button in another group, group four, to choose the type of plot to be used.

This document only reviews a few widgets, but there are many more that can be implemented, there is plenty of documentation to be reviewed. The aim of this application was to integrate a menu with options and different widgets.

The following is the entire code for the application described here.

Table 7. Code for a Grupo Box, a Graphic and a Graph with Parameters

Download code from: https://github.com/saoderivera/PyQt5Tutorial/blob/master/TutorialPyQt507.py

```
'''
In the application we will implement controls and triggers
the controls that oversee here are:
    Implementing a Group Box
    Implementing a Graph
    Implementing a Graph with parameters
'''

import sys
from PyQt5.QtWidgets import QMainWindow, QAction, QMenu, QApplication

from PyQt5.QtWidgets import QSizePolicy

from PyQt5.QtWidgets import QCheckBox    # checkbox
from PyQt5.QtWidgets import QPushButton  # pushbutton
from PyQt5.QtWidgets import QLineEdit    # Lineedit
from PyQt5.QtWidgets import QRadioButton # Radio Buttons
from PyQt5.QtWidgets import QGroupBox    # Group Box


# These components are essential for creating the graphics in pqt5
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure     # Figure
#---------------------------------------------------------------------

from numpy.polynomial.polynomial import polyfit
import numpy as np
```

```
#-------------------------------------------------------------------------
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QMessageBox

from PyQt5.QtCore import pyqtSlot
from PyQt5.QtCore import pyqtSignal
from PyQt5.QtCore import Qt  # Control status
from PyQt5.QtWidgets import  QWidget,QLabel, QVBoxLayout, QHBoxLayout, QGridLayout

#::-------------------------------------------------------------------------------------
---
#:: Class: Group Box
#::-------------------------------------------------------------------------------------
---
class GroupBoxClass(QMainWindow):
    send_fig = pyqtSignal(str)  # To manage the signals PyQT manages the
communication

    def __init__(self):
        #::-------------------------------------------------------------
        # Initialize the values of the class
        # Here the class inherits all the attributes and methods from the
QMainWindow
        #::-------------------------------------------------------------
        super(GroupBoxClass, self).__init__()

        self.Title = 'Title : Group Box '
        self.initUi()

    def initUi(self):
        #::-----------------------------------------------------------------
        #  We create the type of layout QVBoxLayout (Vertical Layout )
        #  This type of layout comes from QWidget
        #::-----------------------------------------------------------------
        self.setWindowTitle(self.Title)
        self.main_widget = QWidget(self)
        self.layout = QVBoxLayout(self.main_widget)    # Creates vertical layout

        self.groupBox1 = QGroupBox('Features')
        self.groupBox1Layout= QGridLayout()
        self.groupBox1.setLayout(self.groupBox1Layout)


        self.Cbox1 = QCheckBox('Feature 1', self)
        self.Cbox1.setChecked(True)
        self.Cbox2 = QCheckBox('Feature 2', self)
        self.Cbox2.setChecked(True)
        self.groupBox1Layout.addWidget(self.Cbox1, 0, 0)
        self.groupBox1Layout.addWidget(self.Cbox2, 0, 1)

        self.setGeometry(300, 300, 250, 150)

        self.layout.addWidget(self.groupBox1)

        self.setCentralWidget(self.main_widget)        # Creates the window with all
the elements
        self.resize(300, 100)                          # Resize the window

#::-------------------------------------------------------------------------------------
---
#:: Class: Line Edit Control
#::-------------------------------------------------------------------------------------
```

```
---
class GraphicClass(QMainWindow):
    send_fig = pyqtSignal(str)  # To manage the signals PyQT manages the
communication

    def __init__(self):
        #::-------------------------------------------------------------
        # Initialize the values of the class
        # Here the class inherits all the attributes and methods from the
QMainWindow
        #::-------------------------------------------------------------
        super(GraphicClass, self).__init__()

        self.Title = 'Title : Simple Graph '
        self.initUi()

    def initUi(self):
        #::-----------------------------------------------------------------
        #  We create the type of layout QVBoxLayout (Vertical Layout )
        #  This type of layout comes from QWidget
        #::-----------------------------------------------------------------
        self.setWindowTitle(self.Title)
        self.main_widget = QWidget(self)
        self.layout = QVBoxLayout(self.main_widget)    # Creates vertical layout

        #::-----------------------------------------------------------------
        #  Creates the containers for the graphic
        self.fig = Figure()
        self.ax1 = self.fig.add_subplot(111)
        self.canvas = FigureCanvas(self.fig)

        self.canvas.setSizePolicy(QSizePolicy.Expanding,
                                  QSizePolicy.Expanding)

        self.canvas.updateGeometry()

        X_1 = [1,2,3,4,5,6,7,8,9,10]
        y_1 = [8,35,40,1,34,33,2,33,11,14]

        self.ax1.scatter(X_1,y_1)

        vtitle = "Example of graph"
        self.ax1.set_title(vtitle)
        self.ax1.set_xlabel("Label for X")
        self.ax1.set_ylabel("Label for y")
        self.ax1.grid(True)

        self.fig.tight_layout()
        self.fig.canvas.draw_idle()


        self.layout.addWidget(self.canvas)

        self.setGeometry(300, 300, 250, 150)

        self.setCentralWidget(self.main_widget)        # Creates the window with all
the elements
        self.resize(500, 450)                          # Resize the window
        self.show()

#::-------------------------------------------------------------------------------
---
#:: Class: Graphic with Params
```

```
#::-------------------------------------------------------------------------------
---
class GraphWParamsClass(QMainWindow):
    send_fig = pyqtSignal(str)  # To manage the signals PyQT manages the
communication

    def __init__(self):
        #::-------------------------------------------------------
        # Initialize the values of the class
        # Here the class inherits all the attributes and methods from the
QMainWindow
        #::-------------------------------------------------------
        super(GraphWParamsClass, self).__init__()

        self.Title = 'Title : Graphic with Parameters '
        self.initUi()

    def initUi(self):
        #::-----------------------------------------------------------------
        #  We create the type of layout QVBoxLayout (Vertical Layout )
        #  This type of layout comes from QWidget
        #::-----------------------------------------------------------------
        self.vcolor = "Orange"
        self.setWindowTitle(self.Title)
        self.main_widget = QWidget(self)
        self.layout = QVBoxLayout(self.main_widget)    # Creates vertical layout

        # Fist the group boxes are created
        self.groupBox1 = QGroupBox('')
        self.groupBox1Layout= QVBoxLayout()
        self.groupBox1.setLayout(self.groupBox1Layout)

        self.groupBox2 = QGroupBox('Color or Regression Line')
        self.groupBox2Layout = QHBoxLayout()
        self.groupBox2.setLayout(self.groupBox2Layout)

        self.groupBox3 = QGroupBox('Graphic')
        self.groupBox3Layout = QVBoxLayout()
        self.groupBox3.setLayout(self.groupBox3Layout)

        # the checkline is created to be added to the first group
        self.chkline = QCheckBox("Include Regression LIne", self)
        self.chkline.setChecked(True)
        self.chkline.stateChanged.connect(self.onClicked)

        self.groupBox1Layout.addWidget(self.chkline)

        # Radio buttons are create to be added to the second group

        self.b1 = QRadioButton("Orange")
        self.b1.setChecked(True)
        self.b1.toggled.connect(self.onClicked)

        self.b2 = QRadioButton("Blue")
        self.b2.toggled.connect(self.onClicked)

        self.b3 = QRadioButton("Red")
        self.b3.toggled.connect(self.onClicked)

        self.buttonlabel = QLabel(self.vcolor+' is selected')

        self.groupBox2Layout.addWidget(self.b1)
        self.groupBox2Layout.addWidget(self.b2)
```

```
        self.groupBox2Layout.addWidget(self.b3)
        self.groupBox2Layout.addWidget(self.buttonlabel)

        # Information to be displayed in the graph
        self.X_1 = np.array( [1,2,3,4,5,6,7,8,9,10])
        self.y_1 = np.array([15,25,30,20,50,55,60,55,70,75])

        # Parameters for the regression line
        self.b, self.m = polyfit(self.X_1, self.y_1, 1)

        # figure and canvas figure to draw the graph is created to

        self.fig = Figure()
        self.ax1 = self.fig.add_subplot(111)
        self.canvas = FigureCanvas(self.fig)

        self.canvas.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

        self.canvas.updateGeometry()

        # Canvas is added to the third group box
        self.groupBox3Layout.addWidget(self.canvas)


        # Adding to the main layout the groupboxes

        self.layout.addWidget(self.groupBox1)
        self.layout.addWidget(self.groupBox2)
        self.layout.addWidget(self.groupBox3)

        self.setCentralWidget(self.main_widget)        # Creates the window with all
the elements
        self.resize(600, 500)                          # Resize the window
        self.onClicked()


    def onClicked(self):

        # Figure is cleared to create the new graph with the choosen parameters
        self.ax1.clear()

        # the buttons are inspect to indicate which one is checked.
        # vcolor is assigned the chosen color
        if self.b1.isChecked():
            self.vcolor = self.b1.text()
        if self.b2.isChecked():
            self.vcolor = self.b2.text()
        if self.b3.isChecked():
            self.vcolor = self.b3.text()

        # the label that displays the selected option
        self.buttonlabel.setText(self.vcolor+' is selected')

        # create the scatter plot , a radio button option could be created
        # to choose a scatter plot or other type of graphic

        self.ax1.scatter(self.X_1, self.y_1)

        # if checkbox for showing regression line is checked
        if self.chkline.isChecked():
            self.ax1.plot(self.X_1, self.b + self.m * self.X_1, '-',
color=self.vcolor)
```

```
        vtitle = "Example of graph with parameters"
        self.ax1.set_title(vtitle)
        self.ax1.set_xlabel("Label for X")
        self.ax1.set_ylabel("Label for y")
        self.ax1.grid(True)

        # show the plot
        self.fig.tight_layout()
        self.fig.canvas.draw_idle()


#::-------------------------------------------------------------
#:: Definition of a Class for the main manu in the application
#::-------------------------------------------------------------
class Menu(QMainWindow):

    def __init__(self):

        super().__init__()
        #::-----------------------
        #:: variables use to set the size of the window that contains the menu
        #::-----------------------
        self.left = 100
        self.top = 100
        self.width = 500
        self.height = 300

        #:: Title for the application

        self.Title = 'Demo PyQt5 - No.7 : Groups and Graphics'

        #:: The initUi is call to create all the necessary elements for the menu

        self.initUI()

    def initUI(self):

        #::-------------------------------------------------
        # Creates the manu and the items
        #::-------------------------------------------------
        self.setWindowTitle(self.Title)
        self.setGeometry(self.left, self.top, self.width, self.height)
        self.statusBar()
        #::-----------------------------
        # 1. Create the menu bar
        # 2. Create an item in the menu bar
        # 3. Creaate an action to be executed the option in the  menu bar is choosen
        #::-----------------------------
        mainMenu = self.menuBar()
        fileMenu = mainMenu.addMenu('File')

        #:: Add another option to the Menu Bar

        exampleWin = mainMenu.addMenu ('Graphics')

        #::-------------------------------------
        # Exit action
        # The following code creates the the da Exit Action along
        # with all the characteristics associated with the action
        # The Icon, a shortcut , the status tip that would appear in the window
        # and the action
        #  triggered.connect will indicate what is to be done when the item in
        # the menu is selected
```

```
        # These definitions are not available until the button is assigned
        # to the menu
        #::-------------------------------------

        exitButton = QAction(QIcon('enter.png'), '&Exit', self)
        exitButton.setShortcut('Ctrl+Q')
        exitButton.setStatusTip('Exit application')
        exitButton.triggered.connect(self.close)

        #:: This line adds the button (item element ) to the menu

        fileMenu.addAction(exitButton)

        #::-------------------------------------------------------------
        #:: Add code to include a Checkbox
        #::-------------------------------------------------------------

        exampleGroupBox = QAction("Group Box", self)
        exampleGroupBox.setStatusTip("Example of Group Box")
        exampleGroupBox.triggered.connect(self.ExampleGroupBox)

        #:: We add the exampleCheckControl to the menu examples
        exampleWin.addAction(exampleGroupBox)

        #::-------------------------------------------------------------
        #:: Add code to include Text Line and button to implement an action upon
request
        #::-------------------------------------------------------------

        exampleGraphic =  QAction("Graphic", self)
        exampleGraphic.setStatusTip('Example of Graphic')
        exampleGraphic.triggered.connect(self.ExampleGraphic)

        exampleWin.addAction(exampleGraphic)

        #::-------------------------------------------------------------
        #:: Add code to include radio buttons  to implement an action upon request
        #::-------------------------------------------------------------

        exampleGWParams =  QAction("Graphic with Parameters ", self)
        exampleGWParams.setStatusTip('Example of Graphic with parameters')
        exampleGWParams.triggered.connect(self.ExampleGraphWParams)

        exampleWin.addAction(exampleGWParams)

        #:: Creates an empty list of dialogs to keep track of
        #:: all the iterations

        self.dialogs = list()

        #:: This line shows the windows
        self.show()


    def ExampleGroupBox(self):
        dialog = GroupBoxClass()
        self.dialogs.append(dialog)  # Appends to the list of dialogs
        dialog.show()      # Show the window

    def ExampleGraphic(self):
        dialog = GraphicClass()
        self.dialogs.append(dialog) # Apppends to the list of dialogs
        dialog.show()
```

```
    def ExampleGraphWParams(self):
        dialog = GraphWParamsClass()
        self.dialogs.append(dialog) # Apppends to the list of dialogs
        dialog.show()

#::------------------------
#:: Application starts here
#::------------------------

def main():
    app = QApplication(sys.argv)  # creates the PyQt5 application
    mn = Menu()  # Cretes the menu
    sys.exit(app.exec_())  # Close the application

if __name__ == '__main__':
    main()
```

The diagram below shows the organization of the methods used in this application.

Diagram 7. Classes and Methods for Section Seven