PyQT tutorial

A tutorial by Eduard Shkulipa

Contents

PyQT is a set of Python bindings for the Qt application framework, enabling the development of cross-platform applications with a rich set of widgets, customizable look and feel, and robust event handling through a signal and slot mechanism. It supports integration with databases, rich text, multimedia, and advanced graphics, making it ideal for desktop applications, scientific tools, business software, educational programs, and rapid prototyping. PyQT's versatility and ease of use allow developers to create visually appealing and functional GUIs efficiently.

Installation

To install PyQT, open terminal and execute the following command:

```
pip install pyqt5 qtpy
```

If you are using MAC, you need to install the package to homebrew and use homebrew python to use the package.

Execute the following:

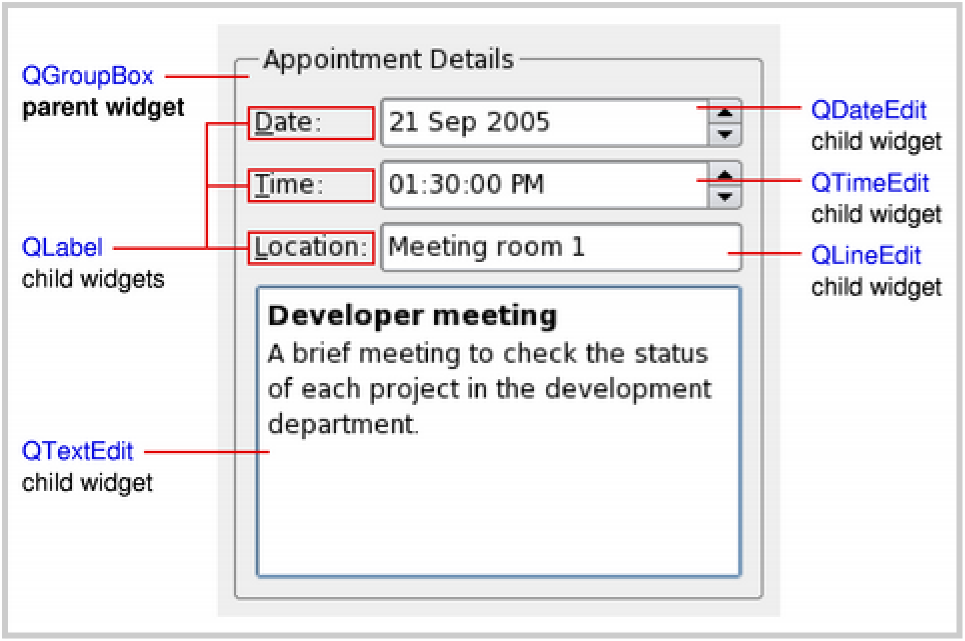
```
brew install PyQt5
```

And then check that the python you are using is homebrew python

```
   $ which python3
/opt/homebrew/bin/python3
```

Structure of the application

PyQt is structured a lot like a webpage, the concepts of CSS, blocks, events, and containers are virtually the same. The overall program is called an **application**, it encapsulates all of the windows, layouts, styles, animations, and so on. A window is called a **widget**. A widget is the term used for the window that we see when the application is executed. A widget contains a **layout**. A layout is similar to a page of the app - it contains different elements like labels, buttons, text entries, pictures, and so on. Those elements are also referred to in the library as **widgets**. In order to eliminate confusion, the parent window that is called a Widget, will be referred to as window, and the elements of a layout will be referred to as widgets. A reference to the official documentation with a more precise definition of a widget and its containers is here



An example structure of a widget, source - QtDocumentation

There can be multiple layouts and a layout can have different contents. In addition, it is possible to add elements to layout, change the contents of the widget, add style to the window, layout, or widgets, and add animations for the elements.

A window is written as a class in the original library, and while it is possible to write programs functionally, creating and adding labels with general functions, it is a more common practice to encapsulate a window and all the different layouts in objects, where a parent is the window, and the class itself has all the elements of the layout in the class constructor and all the event handlers as methods. Since each widget and layout is an object, encapsulating the entire application into a class reduces the complexity with passing of the elements of the applications.

Creating a simple application

As mentioned above, a simple application consists of a Widget class, that has all the layouts created in the constructor and all the event handlers are methods in the class. Then, in the main function, an application is initialized, a widget object is created and shown, and then the application is executed.

```
import sys
import os
from qtpy.QtWidgets import (QApplication, QLabel, QWidget,
                           QVBoxLayout, QPushButton, QTextEdit, QHBoxLayout)


class Widget(QWidget):
   # First, we create a widget layouts
   #and assign the parent calss to be the original QWidget
   def __init__(self, parent = None):
       super(Widget,self).__init__(parent)

       helloLavble = QLabel("Hello World, I am a label") # create a text label
       startButton = QPushButton("Press me to start!") # Create a button with a

       self.nameLable = QLabel() # A placeholder lable
       self.nameEntry = QTextEdit() # Create a text entry
       self.nameRequestLable = QLabel("Please enter your name")
       self.nameRequestButton = QPushButton("Press to learn your name")

       self.layoutMain = QHBoxLayout() # create a layout
       self.layoutName = QVBoxLayout() # create a layout,
       """ there are different types of layouts:
       Vertical layout
       Horizontal layout
       Grid layout,
       ...
       Docs: https://doc.qt.io/qt-6/layout.html"""
       self.layoutMain.addWidget(helloLavble)
       self.layoutMain.addWidget(startButton) # populate the layout with the elements

       self.layoutName.addWidget(self.nameLable)
       self.layoutName.addWidget(self.nameRequestLable)
       self.layoutName.addWidget(self.nameEntry)
       self.layoutName.addWidget(self.nameRequestButton)

       # Create an event handler for buttons
       startButton.clicked.connect(self.__startNames)
       self.nameRequestButton.clicked.connect(self.__setName)

       # set the default layout to main
       self.setLayout(self.layoutMain)

       # Set window properties
       self.setWindowTitle('My App')
       self.setGeometry(500, 500, 500, 500)

   # Chages the start window to a name request window
   def __startNames(self):
       self.__clear_layout(self.layoutMain)
       self.setLayout(self.layoutName)

   # Collects text fdrom entry and puts in a lable
   def __setName(self):
       aName = self.nameEntry.toPlainText() # get the text
       self.nameLable.setText("Your name is: "+aName)

   def __clear_layout(self, layout):
       if layout is not None:
           while layout.count():
               child = layout.takeAt(0)
               if child.widget() is not None:
                   child.widget().deleteLater()
           QWidget().setLayout(layout)  # Remove the layout from any widget it's set
to

if __name__ == '__main__':

   app = QApplication([]) # initialize the app
```

```
w = Widget() # initialize the widget
w.show() # show the contents of the widget

# To add style to the window, it is possible to use the methods inside the class
#or apply them using the qss file, similar to css
qss_file = os.path.join(os.path.dirname(__file__), 'style.qss')
with open(qss_file, "r") as f:
    app.setStyleSheet(f.read())


sys.exit(app.exec_()) # run a loop of the application. exit when wiindow closed
```

//Things to add:

- Explanation of the style sheet + style sheet
- Change of the layouts
- How to add animations: class that is a template for ads: name, general label, background color, image - all follow to the window

References

https://python-qt-tutorial.readthedocs.io/en/latest/
https://doc.qt.io/qt-6/qtwidgets-index.html
https://doc.qt.io/qt-6/layout.html