About the Tutorial

PyQt is a GUI widgets toolkit. It is a Python interface for Qt, one of the most powerful, and popular cross-platform GUI library. PyQt is a blend of Python programming language and the Qt library. This introductory tutorial will assist you in creating graphical applications with the help of PyQt.

Audience

This tutorial is designed for software programmers who are keen on learning how to develop graphical applications using PyQt.

Prerequisites

You should have a basic understanding of computer programming terminologies. A basic understanding of Python and any of the programming languages is a plus.

Disclaimer & Copyright

© Copyright 2018 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com.
Table of Contents

About the Tutorial........................................................................................................... i
Audience ......................................................................................................................... i
Prerequisites ................................................................................................................... i
Disclaimer & Copyright................................................................................................. i
Table of Contents .......................................................................................................... ii

1. PYQT — INTRODUCTION .......................................................................................... 1
2. PYQT — HELLO WORLD ............................................................................................ 3
3. PYQT — MAJOR CLASSES .......................................................................................... 4
4. PYQT — USING QT DESIGNER .................................................................................. 8
5. PYQT — SIGNALS AND SLOTS ................................................................................. 11
6. PYQT — LAYOUT MANAGERS .................................................................................. 14
7. PYQT — QBOXLAYOUT CLASS ............................................................................... 16
8. PYQT — QGRIDLAYOUT CLASS .............................................................................. 20
9. PYQT — QFORMLAYOUT CLASS ........................................................................... 22
10. PYQT — QLABEL WIDGET ...................................................................................... 24
11. PYQT — QLINEEDIT WIDGET ................................................................................ 27
12. PYQT — QPUSHBUTTON WIDGET .......................................................................... 31
13. PYQT — QRADIOBUTTON WIDGET ........................................................................ 35
14. PYQT — QCHECKBOX WIDGET .............................................................................. 38
15. PYQT — QCOMBOBOX WIDGET .............................................................................. 41
37. PyQt — QPIXMAP CLASS ................................................................. 46
38. PyQt — QSCROLLBAR WIDGET ....................................................... 48
39. PyQt — QCALENDAR WIDGET ........................................................ 51
1. PyQt — Introduction

PyQt is a GUI widgets toolkit. It is a Python interface for Qt, one of the most powerful, and popular cross-platform GUI library. PyQt was developed by RiverBank Computing Ltd. The latest version of PyQt can be downloaded from its official website:

www.riverbankcomputing.com/software/pyqt/download

PyQt API is a set of modules containing a large number of classes and functions. While QtCore module contains non-GUI functionality for working with file and directory etc., QtGui module contains all the graphical controls. In addition, there are modules for working with XML (QtXml), SVG (QtSvg), and SQL (QtSql), etc.

Supporting Environments

PyQt is compatible with all the popular operating systems including Windows, Linux, and Mac OS. It is dual licensed, available under GPL as well as commercial license.

Windows

You can download and install an appropriate installer from the above download link corresponding to Python version (2.7 or 3.4) and hardware architecture (32 bit or 64 bit). Note that there are two versions of PyQt that are available namely, PyQt 4.8 and PyQt 5.5.

While PyQt4 is available for Python 2 as well as Python 3, PyQt5 can be used along with Python 3.* only.

| PyQt4 Windows Binaries |  |
|-----------------------|--|---|
| PyQt4-4.11.4-gpl-Py3.4-Qt4.8.7-x64.exe | Windows 64 bit installer |
| PyQt4-4.11.4-gpl-Py3.4-Qt4.8.7-x32.exe | Windows 32 bit installer |
| PyQt4-4.11.4-gpl-Py3.4-Qt5.5.0-x64.exe | Windows 64 bit installer |
| PyQt4-4.11.4-gpl-Py3.4-Qt5.5.0-x32.exe | Windows 32 bit installer |
| PyQt4-4.11.4-gpl-Py2.7-Qt4.8.7-x64.exe | Windows 64 bit installer |
| PyQt4-4.11.4-gpl-Py2.7-Qt4.8.7-x32.exe | Windows 32 bit installer |

| PyQt5 Windows Binaries |  |
|-----------------------|--|---|
| PyQt5-5.5-gpl-Py3.4-Qt5.5.0-x64.exe | Windows 64 bit installer |
| PyQt5-5.5-gpl-Py3.4-Qt5.5.0-x32.exe | Windows 32 bit installer |
**Linux**

For Ubuntu or any other debian Linux distribution, use the following command to install PyQt:

```bash
sudo apt-get install python-qt4
```

or

```bash
sudo apt-get install python-qt5
```

You can also build from the source code available on the ‘download’ page.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PyQt-x11-gpl-4.11.4.tar.gz</td>
<td>Linux, UNIX source for PyQt4</td>
</tr>
<tr>
<td>PyQt-gpl-5.5.tar.gz</td>
<td>Linux, UNIX, MacOS/X source for PyQt5</td>
</tr>
</tbody>
</table>

**Mac OS**

PyQtX project (http://sourceforge.net/projects/pyqtx/) hosts binaries of PyQt for Mac. Use Homebrew installer as per the following command:

```bash
brew install pyqt
```
Creating a simple GUI application using PyQt involves the following steps:

- Import QtGui module.
- Create an application object.
- A QWidget object creates top level window. Add QLabel object in it.
- Set the caption of label as "hello world".
- Define the size and position of window by setGeometry() method.
- Enter the mainloop of application by \texttt{app.exec()} method.

```python
import sys
from PyQt4 import QtGui

def window():
    app = QtGui.QApplication(sys.argv)
    w = QtGui.QWidget()
    b = QtGui.QLabel(w)
    b.setText("Hello World!")
    w.setGeometry(100,100,200,50)
    b.move(50,20)
    w.setWindowTitle("PyQt")
    w.show()
    sys.exit(app.exec_())

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

The above code produces the following output:

![PyQt Window](image)

Hello World!
**PyQt API** is a large collection of classes and methods. These classes are defined in more than 20 modules. Following are some of the frequently used modules:

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QtCore</td>
<td>Core non-GUI classes used by other modules</td>
</tr>
<tr>
<td>QtGui</td>
<td>Graphical user interface components</td>
</tr>
<tr>
<td>QtMultimedia</td>
<td>Classes for low-level multimedia programming</td>
</tr>
<tr>
<td>QtNetwork</td>
<td>Classes for network programming</td>
</tr>
<tr>
<td>QtOpenGL</td>
<td>OpenGL support classes</td>
</tr>
<tr>
<td>QtScript</td>
<td>Classes for evaluating Qt Scripts</td>
</tr>
<tr>
<td>QtSql</td>
<td>Classes for database integration using SQL</td>
</tr>
<tr>
<td>QtSvg</td>
<td>Classes for displaying the contents of SVG files</td>
</tr>
<tr>
<td>QtWebKit</td>
<td>Classes for rendering and editing HTML</td>
</tr>
<tr>
<td>QtXml</td>
<td>Classes for handling XML</td>
</tr>
<tr>
<td>QtAssistant</td>
<td>Support for online help</td>
</tr>
<tr>
<td>QtDesigner</td>
<td>Classes for extending Qt Designer</td>
</tr>
</tbody>
</table>

PyQt API contains more than 400 classes. The **QObject** class is at the top of class hierarchy. It is the base class of all Qt objects. Additionally, **QPaintDevice** class is the base class for all objects that can be painted.

**QApplication** class manages the main settings and control flow of a GUI application. It contains main event loop inside which events generated by window elements and other sources are processed and dispatched. It also handles system-wide and application-wide settings.

**QWidget** class, derived from QObject and QPaintDevice classes is the base class for all user interface objects. **QDialog** and **QFrame** classes are also derived from QWidget class. They have their own sub-class system.

Following diagrams depict some important classes in their hierarchy.
Here is a select list of frequently used widgets:

<table>
<thead>
<tr>
<th>Widget</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QLabel</td>
<td>Used to display text or image</td>
</tr>
<tr>
<td>QLineEdit</td>
<td>Allows the user to enter one line of text</td>
</tr>
<tr>
<td>QTextEdit</td>
<td>Allows the user to enter multi-line text</td>
</tr>
<tr>
<td>QPushButton</td>
<td>A command button to invoke action</td>
</tr>
<tr>
<td>QRadioButton</td>
<td>Enables to choose one from multiple options</td>
</tr>
<tr>
<td>QCheckBox</td>
<td>Enables choice of more than one options</td>
</tr>
<tr>
<td>QSpinBox</td>
<td>Enables to increase/decrease an integer value</td>
</tr>
<tr>
<td>QScrollBar</td>
<td>Enables to access contents of a widget beyond display aperture</td>
</tr>
<tr>
<td>QSlider</td>
<td>Enables to change the bound value linearly</td>
</tr>
<tr>
<td>QComboBox</td>
<td>Provides a dropdown list of items to select from</td>
</tr>
<tr>
<td>QMenuBar</td>
<td>Horizontal bar holding QMenu objects</td>
</tr>
<tr>
<td>QStatusBar</td>
<td>Usually at bottom of QMainWindow, provides status information.</td>
</tr>
<tr>
<td>QToolBar</td>
<td>Usually at top of QMainWindow or floating. Contains action buttons</td>
</tr>
<tr>
<td>QListView</td>
<td>Provides a selectable list of items in ListMode or IconMode</td>
</tr>
<tr>
<td>QPixmap</td>
<td>Off-screen image representation for display on QLabel or QPushButton object</td>
</tr>
<tr>
<td>QDialog</td>
<td>Modal or modeless window which can return information to parent window</td>
</tr>
</tbody>
</table>

A typical GUI based application’s top level window is created by **QMainWindow** widget object. Some widgets as listed above take their appointed place in this main window, while others are placed in the central widget area using various layout managers.

The following diagram shows the QMainWindow framework:
The PyQt installer comes with a GUI builder tool called **Qt Designer**. Using its simple drag and drop interface, a GUI interface can be quickly built without having to write the code. It is however, not an IDE such as Visual Studio. Hence, Qt Designer does not have the facility to debug and build the application.

Creation of a GUI interface using Qt Designer starts with choosing a top level window for the application.

You can then drag and drop required widgets from the widget box on the left pane. You can also assign value to properties of widget laid on the form.
The designed form is saved as demo.ui. This ui file contains XML representation of widgets and their properties in the design. This design is translated into Python equivalent by using pyuic4 command line utility. This utility is a wrapper for uic module. The usage of pyuic4 is as follows:

```
pyuic4 -x demo.ui -o demo.py
```

In the above command, -x switch adds a small amount of additional code to the generated XML so that it becomes a self-executable standalone application.

```python
if __name__ == "__main__":
    import sys
    app = QtGui.QApplication(sys.argv)
    Dialog = QtGui.QDialog()
    ui = Ui_Dialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```

The resultant python script is executed to show the following dialog box:
The user can input data in input fields but clicking on Add button will not generate any action as it is not associated with any function. Reacting to user-generated response is called as **event handling**.
Unlike a console mode application, which is executed in a sequential manner, a GUI based application is event driven. Functions or methods are executed in response to user’s actions like clicking on a button, selecting an item from a collection or a mouse click etc., called events.

Widgets used to build the GUI interface act as the source of such events. Each PyQt widget, which is derived from QObject class, is designed to emit 'signal' in response to one or more events. The signal on its own does not perform any action. Instead, it is ‘connected’ to a ‘slot’. The slot can be any callable Python function.

In PyQt, connection between a signal and a slot can be achieved in different ways. Following are most commonly used techniques:

```python
QtCore.QObject.connect(widget, QtCore.SIGNAL('signalname'), slot_function)
```

A more convenient way to call a slot_function, when a signal is emitted by a widget is as follows:

```python
widget.signal.connect(slot_function)
```

Suppose if a function is to be called when a button is clicked. Here, the clicked signal is to be connected to a callable function. It can be achieved in any of the following two techniques:

```python
QtCore.QObject.connect(button, QtCore.SIGNAL("clicked()"), slot_function)
```

or

```python
button.clicked.connect(slot_function)
```

**Example**

In the following example, two QPushButton objects (b1 and b2) are added in QDialog window. We want to call functions b1_clicked() and b2_clicked() on clicking b1 and b2 respectively.

When b1 is clicked, the clicked() signal is connected to b1_clicked() function

```python
b1.clicked.connect(b1_clicked())
```

When b2 is clicked, the clicked() signal is connected to b2_clicked() function

```python
QObject.connect(b2, SIGNAL("clicked()"), b2_clicked)
```
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *

def window():
    app = QApplication(sys.argv)
    win = QDialog()
    b1 = QPushButton(win)
    b1.setText("Button1")
    b1.move(50,20)
    b1.clicked.connect(b1_clicked)

    b2 = QPushButton(win)
    b2.setText("Button2")
    b2.move(50,50)
    QObject.connect(b2,SIGNAL("clicked()"),b2_clicked)

    win.setGeometry(100,100,200,100)
    win.setWindowTitle("PyQt")
    win.show()
    sys.exit(app.exec_())

def b1_clicked():
    print "Button 1 clicked"

def b2_clicked():
    print "Button 2 clicked"

if __name__ == '__main__':
    window()
Output:
Button 1 clicked
Button 2 clicked
A GUI widget can be placed inside the container window by specifying its absolute coordinates measured in pixels. The coordinates are relative to the dimensions of the window defined by setGeometry() method.

**setGeometry() syntax:**

```python
QWidget.setGeometry(xpos, ypos, width, height)
```

In the following code snippet, the top level window of 300 by 200 pixels dimensions is displayed at position (10, 10) on the monitor.

```python
import sys
from PyQt4.QtGui import *

def window():
    app = QApplication(sys.argv)
    w = QWidget()
    b = QPushButton(w)
    b.setText("Hello World!")
    b.move(50,20)
    w.setGeometry(10,10,300,200)
    w.setWindowTitle("PyQt")
    w.show()
    sys.exit(app.exec_())

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

A **PushButton** widget is added in the window and placed at a position 50 pixels towards right and 20 pixels below the top left position of the window.

This **Absolute Positioning**, however, is not suitable because of following reasons:

- The position of the widget does not change even if the window is resized.
- The appearance may not be uniform on different display devices with different resolutions.
- Modification in the layout is difficult as it may need redesigning the entire form.
PyQt API provides layout classes for more elegant management of positioning of widgets inside the container. The advantages of Layout managers over absolute positioning are:

- Widgets inside the window are automatically resized.
- Ensures uniform appearance on display devices with different resolutions.
- Adding or removing widget dynamically is possible without having to redesign.

QLayout class is the base class from which QBoxLayout, QGridLayout and QFormLayout classes are derived.
End of ebook preview
If you liked what you saw...
Buy it from our store @ https://store.tutorialspoint.com