

PyQt5 Basics Tutorial

By Sam Poirier / darthmorf

Contents

Introduction	2
Installing PyQt5	2
PyQt Test App – Hardcoding UI.....	3

Introduction

PyQt5 is, in my opinion a much better GUI module than the python default; tkinter, however it does require a much greater programming knowledge and confidence than tkinter. You will likely need to be able to debug silent crashes (due to the C++ backend of PyQt – errors there aren't piped back to Python properly) and have some knowledge of OOP and a basic understanding of lambda functions.

Despite these extra requirements, it does allow for a much more professional looking project and is much more powerful.

For this guide I will be using python 3.7 but it should work for most versions of Python after 3.2.

There are two sites that have PyQt related docs on them:

<http://pyqt.sourceforge.net/Docs/PyQt5/> and

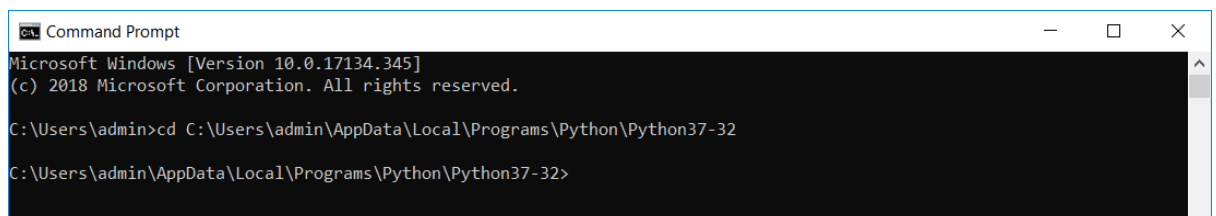
<https://doc.qt.io/qt-5/reference-overview.html>

The second is a much better site but is for the C++ version (Qt) so requires you to translate the code there into Python. Make sure for both that you're on the PyQt5 version!

If you have any suggestions for this tutorial feel free to make an issue.

Installing PyQt5

1. Navigate to the python install directory. Mine is located at:
'C:\Users\admin\AppData\Local\Programs\Python\Python37-32'
There are multiple possible locations python could be installed to, so make sure yours is correct.
2. Open a command prompt and cd to your python install directory:
`cd <your python path>`

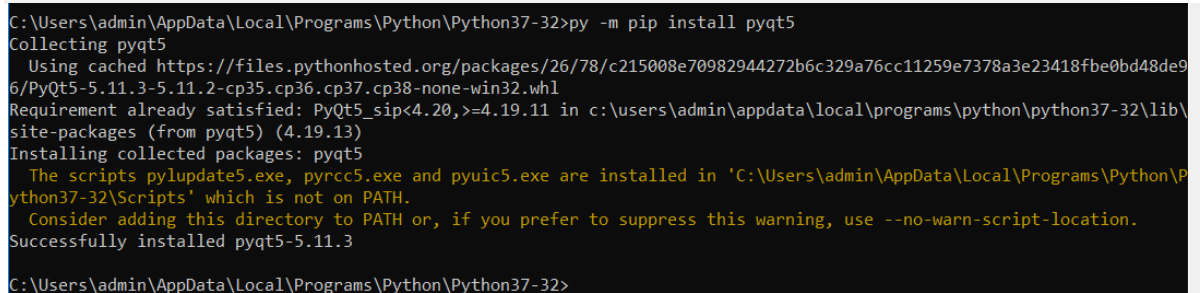


```
Command Prompt
Microsoft Windows [Version 10.0.17134.345]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\admin>cd C:\Users\admin\AppData\Local\Programs\Python\Python37-32
C:\Users\admin\AppData\Local\Programs\Python\Python37-32>
```

3. Use pip to install PyQt5:

```
py -m pip install pyqt5
```



```
C:\Users\admin\AppData\Local\Programs\Python\Python37-32>py -m pip install pyqt5
Collecting pyqt5
  Using cached https://files.pythonhosted.org/packages/26/78/c215008e70982944272b6c329a76cc11259e7378a3e23418fbd0bd48de96/PyQt5-5.11.3-5.11.2-cp35.cp36.cp37.cp38-none-win32.whl
Requirement already satisfied: PyQt5_sip<4.20,>=4.19.11 in c:\users\admin\appdata\local\programs\python\python37-32\lib\site-packages (from pyqt5) (4.19.13)
Installing collected packages: pyqt5
  The scripts pylupdate5.exe, pyrcc5.exe and pyuic5.exe are installed in 'C:\Users\admin\AppData\Local\Programs\Python\Python37-32\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed pyqt5-5.11.3

C:\Users\admin\AppData\Local\Programs\Python\Python37-32>
```

The output should look something like this – the important part is the line 'Successfully installed pyqt5-x.x.x'

4. You can test that PyQt has been installed correctly by quickly booting up a python shell and attempting to import it. Enter:

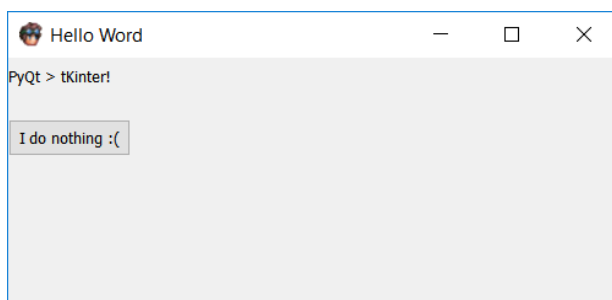
```
python
then
import PyQt5 (once the shell opens)
```

```
C:\Users\admin\AppData\Local\Programs\Python\Python37-32>python
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import PyQt5
>>>
```

If you get no errors, it is installed correctly!

PyQt Test App – Hardcoding UI

There are two main ways to create GUIs in PyQt – the first is to manually program in each element, and the second is to load an xml file containing the layout of the window. For this first example, we will start with the first method.



```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import * # Import all PyQt5 Widgets

class MainWindow(QMainWindow): # Create a 'mainWindow' class that inherits from
the PyQt equivalent
    def __init__(self, *args):
        super().__init__(*args) # Pass the window arguments to the Qt Class - we
don't need it
        self.resize(500, 200) # Set the window size
        self.setWindowTitle("Hello Word")
        self.setWindowIcon(QIcon("icon.png")) # Load an icon from the same
directory

        self.label = QLabel("PyQt > tKinter!", self) # Create a new label element
        self.button = QPushButton("I do nothing :( ", self) # Create a new button
        self.button.move(0, 50) # Move the button 50 pixels down
        self.show() # Show the window

app = QApplication(sys.argv) # Create a PyQt app
mainWindow = MainWindow() # Initialise the MainWindow
sys.exit(app.exec_()) # Wait for the app to close, then close the program
```

This code contains a class inheriting from the PyQt5 QMainWindow class with its own button and label – more types can be found on the docs. The window can be resized and acts and looks like a proper window. It has an icon. It doesn't do anything yet, however. Let's make the label change when we press the button!

The first thing is to create a simple function to update the text of the label. We'll take the new label text as a parameter, and update the button text too:

```
def updateLabelText(self, text):
    self.label.setText(text)
    self.button.setText("I did stuff :D")
```

We also need to connect the signal we get from the button clicked to this 'updateLabelText' function. We will pass the 'button.clicked.connect' function a lambda function which calls the function we just created above. *(I think I said 'function too many times in that sentence').*

```
self.button.clicked.connect(lambda: self.updateLabelText("Button was clicked!"))
```

Our code overall now looks like this:

(See 'TestApp.py')

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import * # Import all PyQt5 Widgets

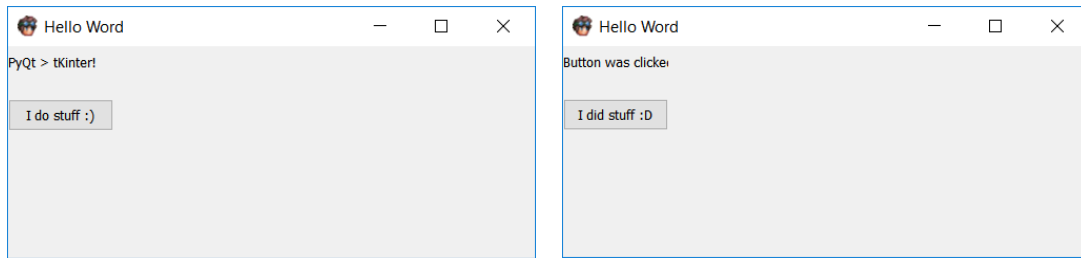
class MainWindow(QMainWindow): # Create a 'mainWindow' class that inherits from
the PyQt equivalent
    def __init__(self, *args):
        super().__init__(*args) # Pass the window arguments to the Qt Class - we
don't need it
        self.resize(500, 200) # Set the window size
        self.setWindowTitle("Hello Word")
        self.setWindowIcon(QIcon("icon.png"))

        self.label = QLabel('PyQt > tKinter!', self) # Create a new label element
        self.button = QPushButton("I do stuff :)", self) # Create a new button
        self.button.move(0, 50) # Move the button 50 pixels down
        self.button.clicked.connect(lambda: self.updateLabelText("Button was
clicked!"))
        self.show() # Show the window

    def updateLabelText(self, text):
        self.label.setText(text)
        self.button.setText("I did stuff :D")

app = QApplication(sys.argv) # Create a PyQt app
mainWindow = MainWindow() # Initialise the MainWindow
sys.exit(app.exec_()) # Wait for the app to close, then close the program
```

When we run this, we are greeted with the first screen, which updates to the second once the button is clicked:



As you can see, the label does not auto-expand to fit the new size. Perhaps now is a good time to set up the designer to create a more advanced UI.