



PyQt4

張傑帆 Chang, Jie-Fan



NTU CSIE

QT

- Qt（官方發音同cute 發音為 /kju:t/[4]，雖然也俗稱為Q.T.發音為 /kju:ti:/）是一個跨平台的C++應用程式開發框架。
- 廣泛用於開發GUI程式，這種情況下又被稱為部件工具箱。也可用於開發非GUI程式，比如控制台工具和伺服器。
- Qt是自由且開放原始碼的軟體，在GNU較寬鬆公共許可證條款下發布。所有版本都支援廣泛的編譯器，包括GCC的C++編譯器和Visual Studio。



PYQT

- PyQt是Python語言的GUI編程解決方案之一
- 可以用來代替Python內置的Tkinter
- 其它替代者還有PyGTK、wxPython等
- 與Qt一樣，PyQt是一個自由軟體
- PyQt是PyKDE的基礎
- 以 PyQt 寫成的著名軟體，包括 Python IDE Eric 和 Linux 遊戲管理器 djl 等，都是優秀的例子



PYQT組件

- PyQt包含了大約440個類型、超過6000個的函數和方法。
- QtCore、QtGui、QtNetwork、QtOpenGL、QtSql、QtXml、QtWebkit、Phonon、uic



安裝軟體

- [PyQt Class Reference](#)

- [PyQt download](#)

- [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



第一個 GUI 程式

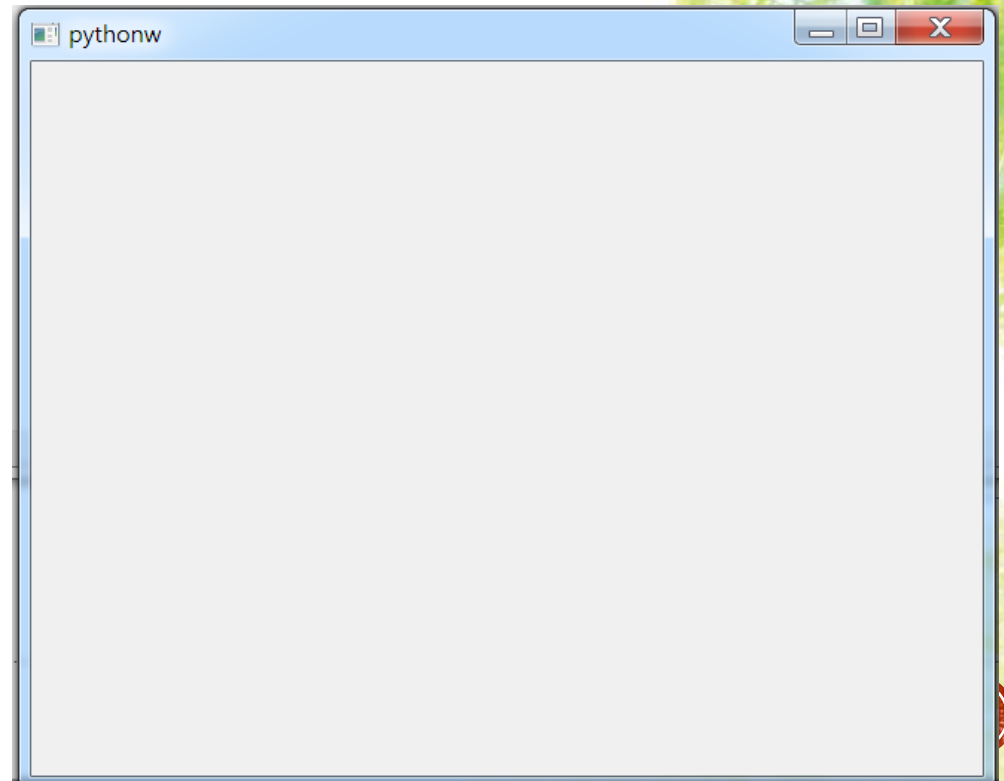
```
import sys
from PyQt4.QtGui import *
```

```
app = QApplication(sys.argv)
```

負責管理 Qt 資源、控制執行流程和其它例行事務

```
widget = QWidget()
widget.show()
```

```
app.exec_()
```



```
#!/usr/bin/env python
```

```
import sys
```

```
from PyQt4.QtGui import *
```

```
app = QApplication(sys.argv)
```

```
widget = QWidget()
```

```
widget.resize(200, 150)
```

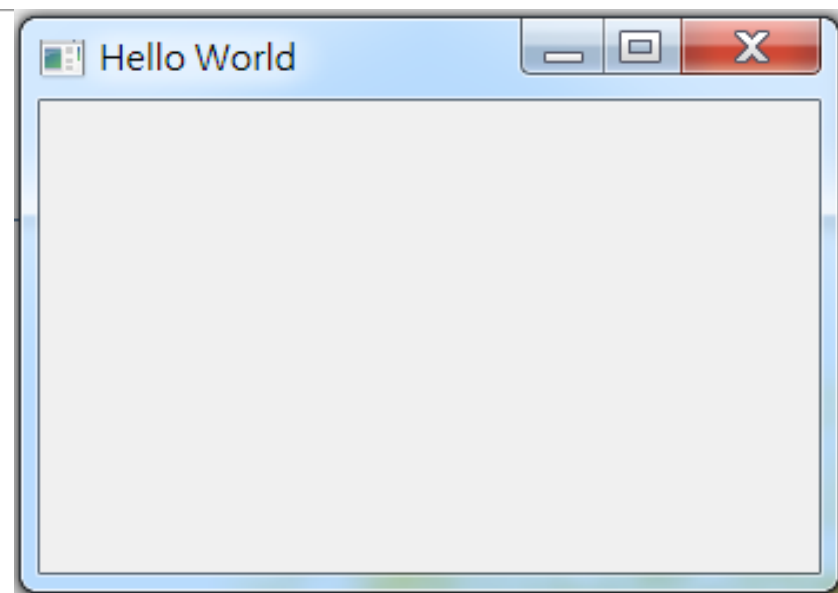
```
widget.move(500, 400)
```

```
widget.setWindowTitle("Hello World")
```

```
widget.show()
```

```
app.exec_()
```

- 只多了 3 行
- 調用 **QWidget instance** 的一些 **memfunc** 改變視窗狀態
- 包括了改變**視窗大小**、把**視窗移動到螢幕特定位置**（而不像初體驗一樣，任由作業系統分派），以及**改變視窗的 title**



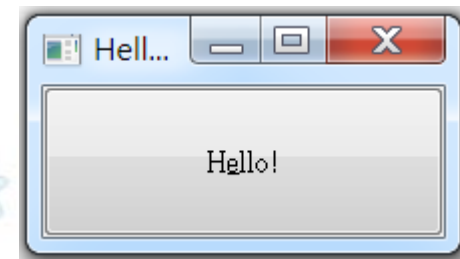

```
#!/usr/bin/env python
```

```
import sys
from PyQt4.QtGui import *

app = QApplication(sys.argv)

button = QPushButton("H&ello!")
button.resize(200, 75)
button.move(500, 400)
button.setWindowTitle("Hello World")
button.show()

app.exec_()
```



- 主角從 **QWidget** 變成 **QPushButton**
- 它帶一個字串參數初始化顯示的文字，並調用一些從 **QWidget** 繼承來的 **function**
- 注意字串 **e** 下面那條底線；試著按按看 **Alt + E**。
- 這是 Qt 設定 **action focus** 的簡便方法。



排版系統

- **QHBoxLayout** 和 **QVBoxLayout**
掌管基本排版：
- 就像裝箱一樣
用 **addWidget()**
依序塞入 **widget**
再將它設為空容器
QWidget 的 **layout**
- 被裝箱的 **widget**
不必再額外調用 **show()**

```
#!/usr/bin/env python
```

```
import sys
```

```
from PyQt4.QtGui import *
```

```
app = QApplication(sys.argv)
```

```
ok = QPushButton("&OK")
```

```
cancel = QPushButton("&Cancel")
```

```
layout = QHBoxLayout()
```

```
layout.addWidget(ok)
```

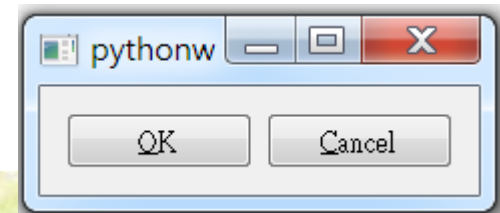
```
layout.addWidget(cancel)
```

```
widget = QWidget()
```

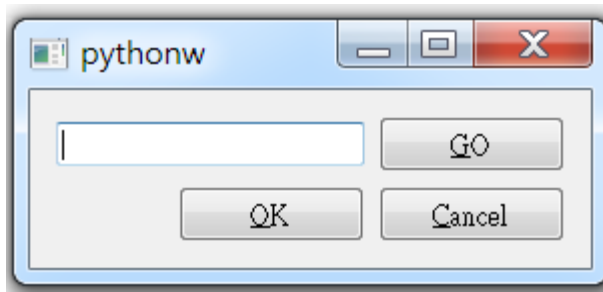
```
widget.setLayout(layout)
```

```
widget.show()
```

```
app.exec_()
```



- 多了些 widget 以及 addLayout()
- 像這樣把 layout 塞入 layout，就可以實現層層堆砌的複雜排版。另外還有 addStretch()，塞入空白的可伸縮空間
- 拖拉邊框，可以看到 Qt 怎樣自動調整 widget 們的大小並管理定位。
- 一般來說，交給 Qt 內部 Layout 系統 來排列 widget 就夠了，很是省心
- 其它排版用的 layout class 還有 QGridLayout、QFormLayout 等，和一堆 container class
- 有興趣的話再去查文檔吧！



```
#!/usr/bin/env python
```

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

```
app = QApplication(sys.argv)
```

```
lineEdit = QLineEdit()
go = QPushButton("&GO")
h1 = QHBoxLayout()
h1.addWidget(lineEdit)
h1.addWidget(go)
```

```
ok = QPushButton("&OK")
cancel = QPushButton("&Cancel")
```

```
h2 = QHBoxLayout()
h2.addStretch(1)
h2.addWidget(ok)
h2.addWidget(cancel)
```

```
layout = QVBoxLayout()
layout.addLayout(h1)
layout.addLayout(h2)
```

```
widget = QWidget()
widget.setLayout(layout)
widget.show()
```

```
app.exec_()
```



- 回到先前的範例，一個簡單的 **PushButton**：

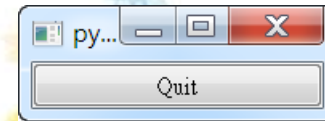
```
#!/usr/bin/env python
```

```
import sys
from PyQt4.QtGui import *

app = QApplication(sys.argv)

button = QPushButton("&Quit")
button.show()

app.exec_()
```



- 儘管按鈕上寫著大大的「Quit」，點個幾百遍也是無動於衷



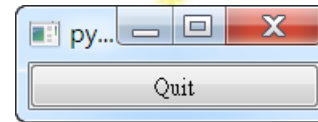
- 為了讓 widget 間能彼此溝通，Qt 有一套 **Signals & Slots** 機制

```
#!/usr/bin/env python
```

```
import sys
from PyQt4.QtGui import *

app = QApplication(sys.argv)

button = QPushButton("&Quit")
button.clicked.connect(button.close)
button.show()
```



```
app.exec_()
```

- 新增的那行：把 button 的 clicked() 這個 signal，與 button 的 close() 這個 slot 聯繫 (connect) 起來
- 之後當 button 被按下 (觸發 clicked()) 時，送出的訊號就會把自己關掉



事件觸發物件.signalName.connect(事件目標物件.slotName)




```
#!/usr/bin/env python
```

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

app = QApplication(sys.argv)

spinBox = QSpinBox()
spinBox.setPrefix("$")
spinBox.setRange(0, 100)

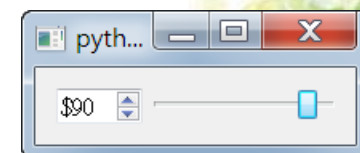
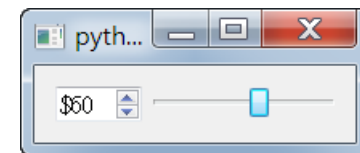
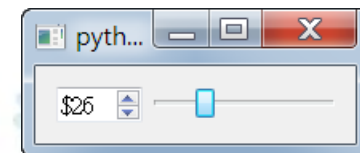
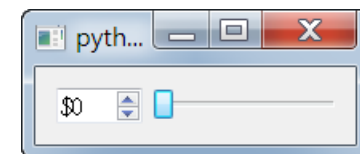
slider = QSlider(Qt.Horizontal)
slider.setRange(0, 100)

spinBox.valueChanged.connect(slider.setValue)
slider.valueChanged.connect(spinBox.setValue)

layout = QHBoxLayout()
layout.addWidget(spinBox)
layout.addWidget(slider)

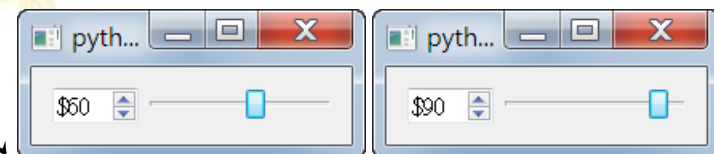
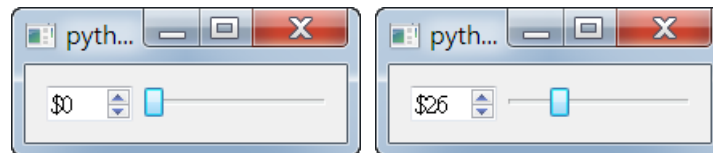
widget = QWidget()
widget.setLayout(layout)
widget.show()

app.exec_()
```



物件導向的寫法

- 將先前那個 **connect** 的例子改成 **class** 型式



```
#!/usr/bin/env python
```

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

```
class MyWidget(QWidget): #繼承自外觀空白一片的QWidget
```

```
    def __init__(self, parent = None):
```

```
        super(MyWidget, self).__init__(parent)
```

```
        self.createLayout()
```

```
        self.spinBox.valueChanged.connect(self.slider.setValue)
```

```
        self.slider.valueChanged.connect(self.spinBox.setValue)
```

```
    def createLayout(self):
```

```
        self.spinBox = QSpinBox()
```

```
        self.spinBox.setPrefix("$")
```

```
        self.spinBox.setRange(0, 100)
```

```
        self.slider = QSlider(Qt.Horizontal)
```

```
        self.slider.setRange(0, 100)
```

```
        layout = QHBoxLayout()
```

```
        layout.addWidget(self.spinBox)
```

```
        layout.addWidget(self.slider)
```

```
        self.setLayout(layout)
```

這 **class** 不需要寄生在其它 **widget** 中，
所以設定參數 **parent = None**，成為
top-level window

```
app = QApplication(sys.argv)
```

```
widget = MyWidget()
```

```
widget.show()
```

```
app.exec_()
```














示範操作

- 開啟 **Designer** 創建、拖曳需要的控制項
- 存成 **.ui** 檔案
- **pyuic4 -x** 檔名 **.ui** -o 檔名 **.py**



輸入元件

Input Widgets	Buttons
 Combo Box	 Push Button
 Font Combo Box	 Tool Button
 Line Edit	 Radio Button
 Text Edit	 Check Box
 Plain Text Edit	 Command Link Button
	 Dialog Button Box

alignment	置右對齊，垂直置中
水平	置右對齊
垂直	垂直置中

font	A [新細明體, 24]
cursor	I 型
mouseTracking	<input type="checkbox"/>
focusPolicy	StrongFocus
contextMenuPolicy	DefaultContextMenu
acceptDrops	<input checked="" type="checkbox"/>



建立按鈕事件

- 建立按鈕事件
- `QtCore.QObject.connect(self.pushButton_ad`
`d, QtCore.SIGNAL("clicked()"), self.執行函式)`
- Ex:

`#Add`

```
QtCore.QObject.connect(self.btAdd, QtCore.SIGNAL("clicked()"), self.Add)
```

`#Equal`

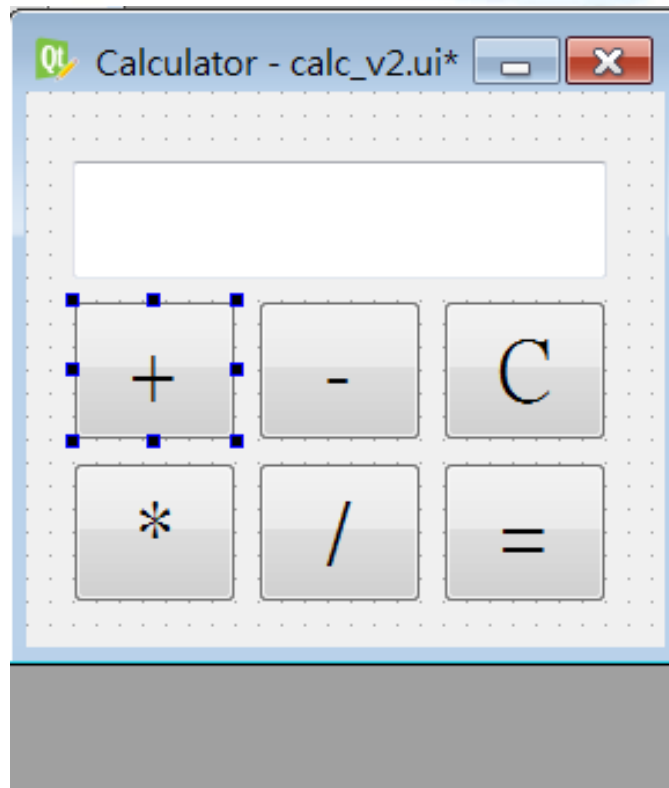
```
QtCore.QObject.connect(self.btEqu, QtCore.SIGNAL("clicked()"), self.Equal)
```

- 建立處理方法(class的函式)

```
def Add(self):  
    self.li.append(self.lineEdit.text())  
    self.li.append("+")  
    calString= ''.join(self.li)  
    self.label.setText(calString)  
    self.lineEdit.setText("")
```

```
def Equal(self):  
    self.li.append(self.lineEdit.text())  
    #最後一個是四則運算  
    if self.lineEdit.text()=="":  
        del self.li[-2]  
    calString= ''.join(self.li)  
    ans=eval(calString)  
    self.label.setText("")  
    self.lineEdit.setText(str(ans))  
    self.li=[]
```

- focus

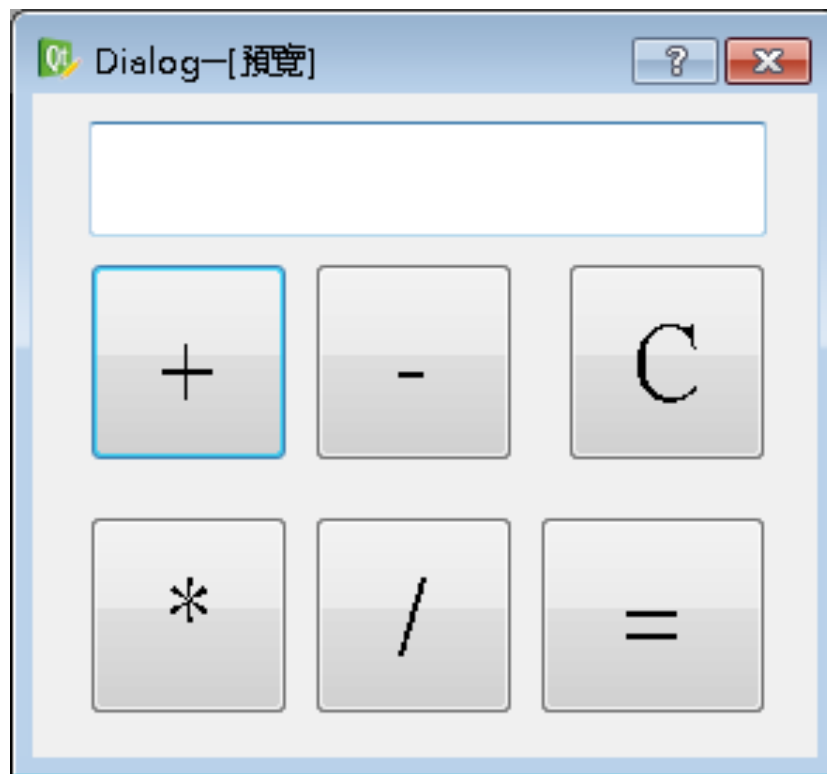


btAdd : QPushButton	
屬性	數值
高度	0
baseSize	0x0
寬度	0
高度	0
palette	繼承
font	A [新細明體, 22]
cursor	鼠 箭頭
mouseTracking	<input type="checkbox"/>
focusPolicy	NoFocus
contextMenuPolicy	DefaultContextMenu
acceptDrops	<input type="checkbox"/>
toolTip	
可翻譯	<input checked="" type="checkbox"/>



作業練習

- 實做一個簡易型的計算機



REFERENCE

- Qt
 - <http://www.qt.io/developers/>
- PyQt4
 - <https://riverbankcomputing.com/software/pyqt/download>
- OGC Gains Comfort
 - <http://ogc-daily.blogspot.tw/>

