https://wikidocs.net/5220

PyQt 를 이용한 GUI 프로그래밍 (revision)

이미 12 장을 통해 간단한 GUI(Graphical User Interface) 프로그래밍을 해봤습니다. 이번 장에서는 앞에서 살펴본 PyQt 내용을 간단히 복습한 후 PyQt 에 대해 좀 더 자세히 살펴보겠습니다.

사실 알고리즘 트레이딩을 하는 데 GUI 프로그래밍이 꼭 필요한 것은 아닙니다. GUI 나 TUI(Text-based User Interface)는 데이터를 효과적으로 표현하기 위한 수단일 뿐 프로그램의 핵심 로직이나 알고리즘과는 관계가 없기 때문입니다. 그렇지만 여러분이 자주 사용하는 HTS 가 텍스트 기반으로 동작한다면 조금 불편하겠지요? 이번 장을 통해 PyQt 를 제대로 배워둔다면 여러분들이 만들려는 프로그램을 쉽게 개발할 수 있을 것입니다.

1. PyQt 기초

PyQt 는 파이썬에서 GUI 프로그래밍을 할 때 사용하는 대표적인 패키지입니다. 현재 사용 중인 아나콘다 배포판(Anaconda 4.3.1)에는 현재 PyQt5 버전이 기본 패키지로 포함돼 있습니다. 이번 절에서는 아나콘다에서 PyQt 를 설치 및 업데이트하는 방법과 12 장에서 다룬 PyQt 의 기초 부분에 대해 간단히 복습해 보겠습니다.

1) 아나콘다에서 PyQt 설치 및 업데이트

PyQt 는 아나콘다 배포판을 설치하면 기본적으로 설치되는 패키지입니다. 따라서 아나콘다 배포판 버전에 따라 설치된 PyQt 패키지 버전이 다를 수 있습니다. 먼저 그림 16.1 과 같이 윈도우의 프로그램 메뉴에서 [Anaconda Prompt]를 클릭해 Anaconda Prompt 를 실행합니다.

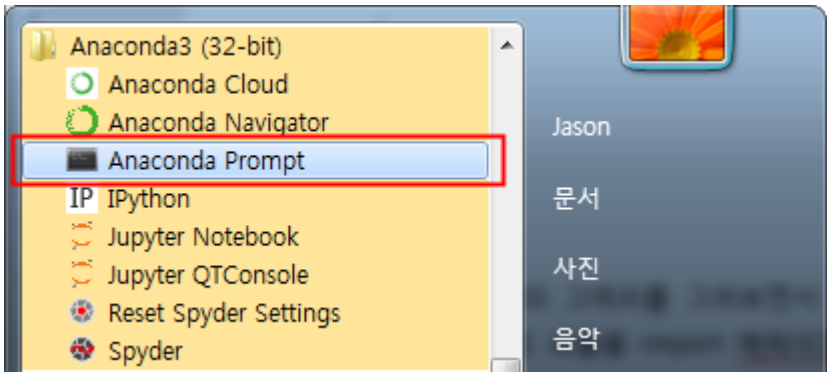


그림 16.1 Anaconda Prompt 실행

아나콘다 배포판에서 현재 설치된 패키지 목록을 확인하려면 conda list 명령을 사용하면 됩니다. 참고로 conda list 명령 이후에 아무것도 입력하지 않으면 배포판에 설치된 모든 패키지의 정보가 출력됩니다.

PyQt 패키지에 대한 정보만 확인하려면 그림 16.2 와 같이 conda list pyqt 를 입력하면 됩니다. 이때 패키지의 이름은 대소문자를 구분하지는 않습니다. 즉, pyqt 라고 입력하거나 PyQt 라고 입력해도 정상적으로 PyQt 패키지 정보를 확인할 수 있습니다.

그림 16.2 conda list pyqt 를 이용한 패키지 정보 확인

그림 16.2 를 참조하면 현재 저자의 개발 환경에서는 파이썬 3.7 로 빌드된 PyQt 5.9.2 버전의 패키지가 설치돼 있음을 확인할 수 있습니다. 아나콘다 배포판을 개발하고 있는 Continuum 에서는 개별 패키지에 대해 지속적으로 업데이트하고 있습니다. 따라서 아나콘다 배포판 사용자는 배포판을 새로 설치하지 않고도 특정 패키지만 업데이트할 수 있습니다.

Anaconda Prompt 에서 패키지를 업데이트하려면 conda update 명령을 사용하면 됩니다. 예를 들어, PyQt 패키지를 업데이트하려면 conda update pyqt 를 입력하면 됩니다. 그림 16.3 을 참조하면 현재는 PyQt 를 업데이트할 내용이 없는 것을 확인할 수 있습니다.

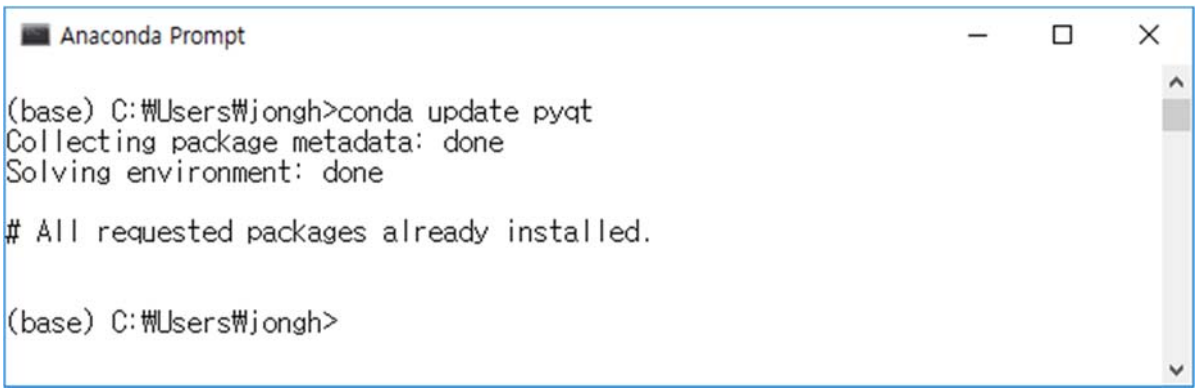


그림 16.3 conda update 를 사용한 PyQt 패키지 업데이트

2) PyQt 복습하기

이번 절에서는 12.2 절에서 설명한 PyQt 기초를 복습하겠습니다. PyQt 가 잘 기억나지 않는다면 잠깐 12.2 절을 먼저 읽어보기 바랍니다.

PyQt 는 따로 한 권 분량의 책이 있을 만큼 복잡하고 규모가 큰 라이브러리입니다. 따라서 이 책에서 PyQt 를 모두 다룰 수는 없습니다. 대신 알고리즘 트레이딩 프로그램 UI 를 만들 수 있을 만큼만 공부하겠습니다.

PyQt 는 분명히 복잡한 라이브러리이긴 하지만 PyQt 기반의 GUI 프로그램의 핵심 구성 요소는 그림 16.4 와 같이 정리할 수 있습니다. 먼저 화면에 출력되는 UI 부분이 있습니다. 이는 PyQt 가 기본적으로 제공하는 위젯(Widget) 클래스의 객체를 생성해서 만들 수 있습니다. 수많은 위젯 중에서 여러분이 사용할 위젯을 선택하고 각 위젯 클래스의 사용법을 익혀야 합니다.

두 번째는 이벤트 루프입니다. PyQt 에서는 QApplication 객체에서 exec_ 메서드를 호출해 이벤트 루프를 생성합니다. 여러분은 이벤트 루프를 생성만 할 뿐 실제로 이벤트 루프에서 여러분이 관여하는 일은 많지 않습니다.

세 번째는 이벤트를 처리할 함수 또는 메서드를 구현하는 부분입니다. 버튼과 같은 위젯을 클릭하면 해당 위젯은 'clicked'라는 시그널(signal)을 발생시킵니다. 여러분은 'clicked'라는 시그널이 발생했을 때 호출되는 함수 또는 메서드를 구현해야 합니다. PyQt 에서는 특정 시그널이 발생했을 때 호출되는 함수 또는 메서드를 슬롯(slot)이라고 합니다. 참고로 다른 프로그래밍 언어에서는 콜백 함수(callback function)라고 부르기도 합니다.

PyQt 에서는 위젯에서 발생하는 시그널에 대해 어떤 슬롯으로 처리할지에 대해 미리 등록함으로써 특정 위젯에서 시그널이 발생했을 때 이벤트 루프가 미리 연결된 슬롯을 자동으로 호출하게 돼 있습니다.

GUI 프로그램의 전반적인 실행 흐름에 대해 정리해 보겠습니다. 그림 16.4 에서 사용자가 버튼 위젯을 클릭하면 'clicked'라는 시그널이 발생합니다. GUI 프로그램의 시작과 함께 생성돼 있던 이벤트 루프는 해당 시그널에 등록된 슬롯을 호출함으로써 사용자가 발생시킨 이벤트를 처리합니다. 이벤트 루프는 사용자가 프로그램을 종료한다는 시그널을 보내기 전까지는 계속해서 이와 같은 방식으로 시그널과 연결된 슬롯을 호출함으로써 이벤트를 처리하는 역할을 합니다.

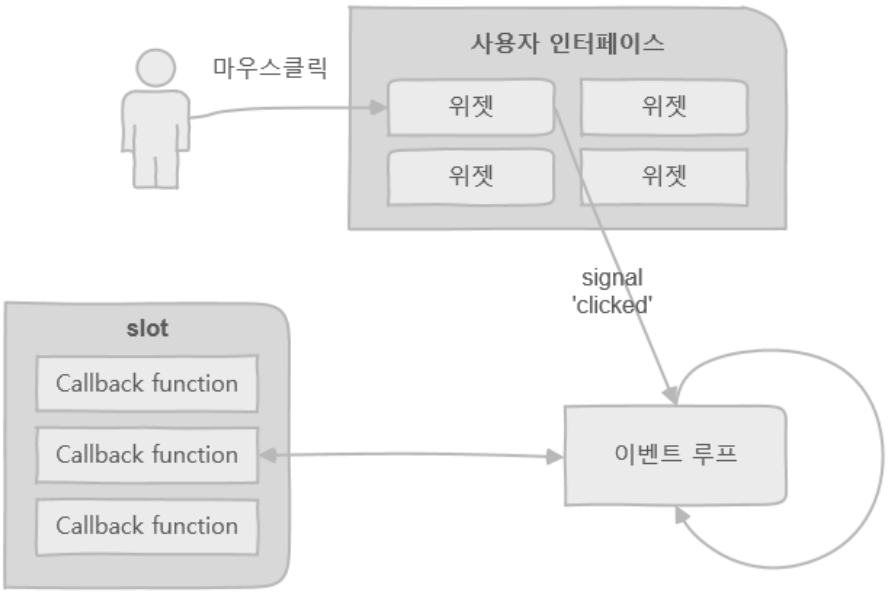


그림 16.4 PyQt 의 UI-시그널-슬롯-이벤트 루프의 관계

PyQt 의 전반적인 동작 방식을 복습했으니, 이번에는 간단한 코드를 통해 이벤트 루프를 생성하는 QApplication 객체를 살펴보겠습니다. 앞서 설명한 것처럼 그림 16.4 의 이벤트 루프는 QApplication 클래스의 객체를 생성한 후 exec_ 메서드를 호출하는 순간 생성됩니다.

한번 생성된 이벤트 루프는 사용자가 윈도우를 닫을 때까지 실행되면서 위젯에서 발생한 시그널을 처리하는 슬롯을 호출하는 역할을 합니다. 이벤트 루프의 생성과 종료 시점을 확인해보기 위해 다음과 같이 코드를 작성해 보겠습니다.

```
import sys
from PyQt5.QtWidgets import *

app = QApplication(sys.argv)
label = QLabel("Hello, PyQt")
label.show()

print("Before event loop")
app.exec_()
print("After event loop")
```

작성된 코드에서는 이벤트 루프를 생성하는 코드의 앞쪽과 뒤쪽에 문자열이 출력되게 했습니다. 코드를 실행하면 그림 16.5 와 같이 QLabel 위젯이 나타나고 콘솔 창에는 'Before event loop'라는 문자열이 출력됩니다.

여기서 한 가지 중요한 점은 파이썬 코드는 파일의 위에서부터 아래쪽으로 순차적으로 실행되는데 app.exec_ 앞쪽의 코드는 실행됐지만 그다음 줄의 코드는 아직 실행되지 않았다는 점입니다. 이는 exec_ 메서드가 호출되면서 이벤트 루프가 생성됐고 이 때문에 프로그램이 계속해서 이벤트 루프 안에서 실행되고 있기 때문입니다.

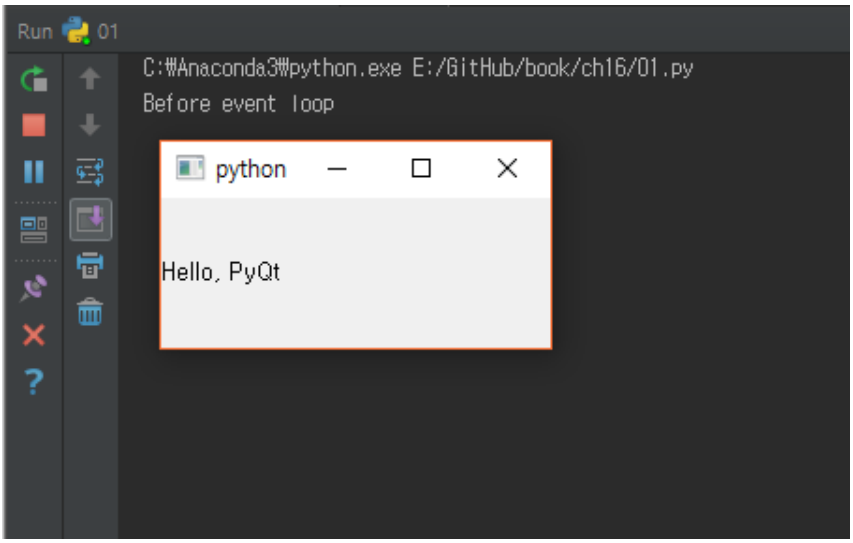


그림 16.5 이벤트 루프 생성(소스 코드: book/ch16/01.py)

이제 화면에 출력된 QLabel 위젯을 닫아 보겠습니다. 윈도우의 'X' 부분을 눌러 QLabel 위젯을 종료하면 exec_ 메서드 호출 시점에 생성됐던 이벤트 루프가 종료되면서 그다음 줄의 코드가 실행됩니다. 따라서 그림 16.6 과 같이 콘솔 창에 'After event loop'라는 문자열이 출력됨을 확인할 수 있습니다.

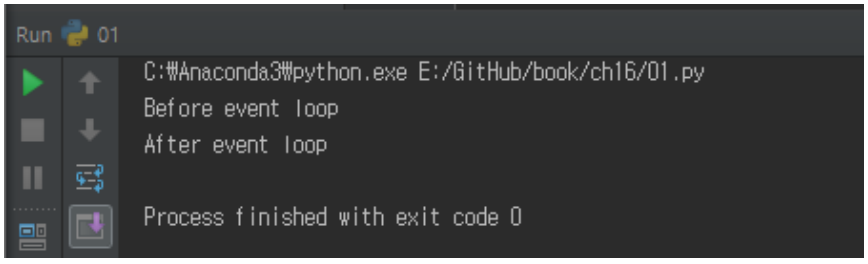


그림 16.6 이벤트 루프 종료

이번에는 시그널과 슬롯을 연결하는 부분에 대한 코드를 작성해 보겠습니다. PyQt 는 위젯의 종류에 따라 발생 가능한 기본 시그널이 정의돼 있습니다. 예를 들어, QPushButton 위젯은 마우스 클릭을 했을 때 'clicked'라는 시그널이 발생합니다.

시그널이 발생했을 때 호출되는 함수 또는 메서드를 슬롯이라 부른다고 했습니다. 슬롯을 구현했다면 시그널과 슬롯을 연결만 해주면 됩니다. 시그널이 발생했을 때 연결된 슬롯을 호출하는 역할은 앞서 살펴본 이벤트 루프가 알아서 처리해줍니다.

다음 코드에서는 QPushButton 위젯을 생성하고 해당 위젯에서 'clicked' 시그널이 발생하면 호출되는 슬롯인 clicked_slot 함수를 구현했고 시그널과 슬롯을 이벤트 루프를 생성하기 전에 미리 연결했습니다.

```
import sys
from PyQt5.QtWidgets import *

def clicked_slot():
    print('clicked')

app = QApplication(sys.argv)

btn = QPushButton("Hello, PyQt")
btn.clicked.connect(clicked_slot)
btn.show()

app.exec_()
```

위 코드를 실행해 보면 그림 16.7 과 같이 QPushButton 객체가 화면에 출력됩니다. 버튼을 클릭하면 콘솔 창에 'clicked'라는 문자열이 출력됨을 확인할 수 있습니다. 이는 QPushButton 위젯에서 'clicked' 이벤트가 발생하면 'clicked_slot'이라는 이름의 함수가 호출되도록 미리 연결해 뒀기 때문입니다.

여기서 한 가지 주의할 점은 슬롯의 구현과 시그널-슬롯의 연결이 모두 이벤트 루프를 생성하는 app.exec_ 메서드 호출보다 먼저 수행돼야 한다는 점입니다.

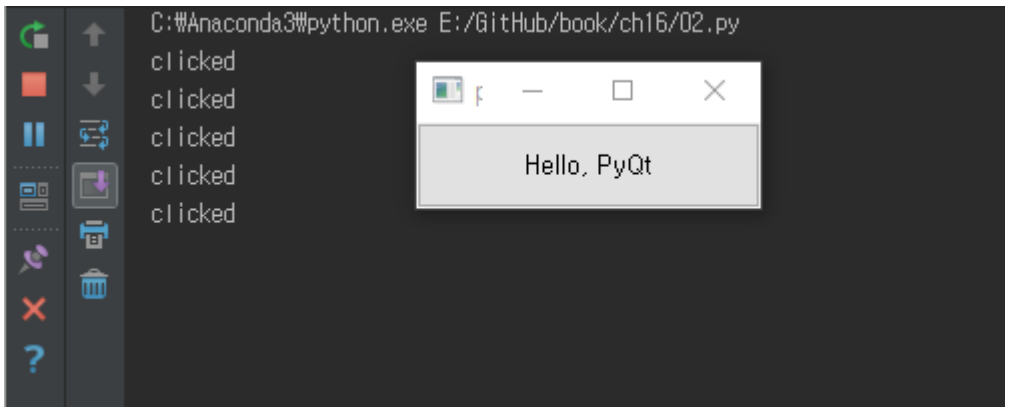


그림 16.7 간단한 시그널-슬롯 연결의 예

지금까지 이벤트 루프 생성과 시그널-슬롯의 연결에 대해 복습했습니다. 이번에는 위젯을 이용해 UI 를 구성하는 부분에 대해 복습하겠습니다. 이번에 구현할 프로그램은 그림 16.8 과 같습니다.

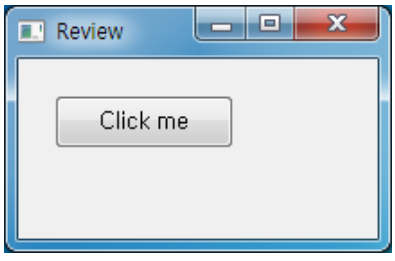


그림 16.8 윈도우 및 이벤트 처리

PyQt 에서 위젯은 UI 를 구성하는 핵심 요소입니다. PyQt 에서는 모든 위젯이 최상위 위젯을 의미하는 윈도우가 될 수 있습니다. 그러나 대부분의 프로그램에서는 QMainWindow 나 QDialog 클래스를 사용해 윈도우를 생성합니다.

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        self.setWindowTitle("Review")

        btn1 = QPushButton("Click me", self)
        btn1.move(20, 20)
        btn1.clicked.connect(self.btn1_clicked)

    def btn1_clicked(self):
        QMessageBox.about(self, "message", "clicked")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

예제 16.1 QMainWindow 를 이용한 윈도우 구성(소스코드: book/ch16/03.py)

그림 16.8 프로그램의 전체 코드는 예제 16.1 과 같습니다. 코드가 기존 코드에 비해 복잡해 보이지만 각 부분을 나눠서 이해하면 쉽게 파악할 수 있습니다. 가장 먼저 살펴볼 부분은 QApplication 객체를 생성하고 이벤트 루프를 만드는 부분입니다.

먼저 QApplication 객체인 app 을 생성하고 exec_ 메서드를 호출해서 이벤트 루프를 생성합니다. 그리고 이벤트 루프를 생성하기 전에 MyWindow 라는 클래스의 객체를 생성하는 것을 확인할 수 있습니다.

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

앞서 배운 내용을 상기해보면 이벤트 루프를 생성하기 전에는 UI 를 구성해야 하고 UI 를 구성하는 기본 단위인 위젯에 대해 시그널과 슬롯을 연결해야 했습니다.

UI 가 간단하다면 몇 줄의 코드만으로도 구현할 수 있겠지만 UI 가 복잡한 경우에는 클래스를 사용하는 것이 좋습니다. 특히 클래스를 사용하면 기존 위젯 클래스를 상속함으로써 여러분만의 클래스를 새롭게 정의할 수 있습니다.

MyWindow 클래스의 객체는 이벤트 루프가 실행되기 전에 생성되며 이때 자동으로 생성자가 호출됩니다. 따라서 코드를 분석할 때는 먼저 생성자 부분부터 살펴보는 것이 좋습니다. 다음 코드를 살펴보면 MyWindow 클래스는 QMainWindow 클래스를 상속받고 있습니다.

자식 클래스인 MyWindow 가 부모 클래스인 QMainWindow 를 상속받으면 QMainWindow 클래스에 정의된 모든 메서드나 프로퍼티를 사용할 수 있습니다. 클래스의 상속은 우리 생활에서 보면 부모님을 잘 만나서 태어날 때부터 '금수저'를 물고 태어나는 것과 비슷합니다.

```
class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
```

```
        self.setupUI()
```

MyWindow 클래스는 단순히 부모 클래스를 상속할 뿐만 아니라 자기 자신의 UI 를 변경하는 메서드인 setupUI 를 호출합니다. 물론 QPushButton 과 같은 객체 생성 코드를 모두 MyWindow 클래스의 생성자에 넣을 수도 있지만, 따로 메서드로 구성해서 코드의 가독성을 좋게 합니다. 여기서 중요한 코드는 'clicked'라는 시그널과 이를 처리하는 슬롯인 self.btn1_clicked 를 연결하는 코드입니다.

```
def setupUI(self):
    self.setWindowTitle("Review")

    btn1 = QPushButton("Click me", self)
    btn1.move(20, 20)
    btn1.clicked.connect(self.btn1_clicked)
```

QPushButton 에서 'clicked' 시그널이 발생할 때 자동으로 호출되는 슬롯인 self.btn1_clicked 는 다음과 같이 MyWindow 클래스의 메서드로 구현합니다.

```
def btn1_clicked(self):
    QMessageBox.about(self, "message", "clicked")
```

2. Qt Designer

지금까지는 UI 구성을 위해 직접 파이썬 코드를 작성했습니다. 간단한 UI 를 구성하는 경우에는 이처럼 직접 코드를 작성하는 것도 편리합니다. 하지만 복잡한 UI 를 구성할 때는 UI 구성을 위한 전용 툴인 Qt Designer 를 사용하는 것이 더 편리합니다.

Qt Designer 는 이름에서 알 수 있듯이 Qt 의 컴포넌트를 이용해 GUI 를 설계하는 전용 툴입니다. 설계라는 단어가 들어 있어 복잡해 보이지만 Qt Designer 는 프로그래머가 GUI 구성을 위지위그(WYSIWYG: What You See Is What You Get) 방식으로 수행할 수 있도록 지원합니다.

1) Qt Designer 소개

앞 절에서 배운 GUI 프로그래밍을 정리해보면 GUI 프로그램의 코드는 다음과 같이 크게 세 부분으로 구성돼 있습니다.

1) GUI 레이아웃
2) 시그널-슬롯 연결 및 슬롯 처리 함수(메서드)
3) 이벤트 루프

위의 세 가지 중 이벤트가 발생했을 때 실제로 프로그램에서 수행해야 하는 작업을 프로그래머가 함수나 메서드로 구현하는 것은 필수적입니다. 이런 것까지 프로그램이 알아서 판단하고 처리한다면 그것은 인공지능 프로그램과 같을 것입니다.

하지만 버튼과 같은 위젯을 생성하고 화면에 출력될 윈도우의 크기를 조절하는 것과 같은 사소한 작업을 모두 파이썬 코드로 구현하는 것은 조금 귀찮고 따분한 작업일지도 모릅니다. 마이크로소프트의 파워포인트에서 도형을 사용해 그림을 그리는 것처럼 코드 한 줄 작성하지 않고도 쉽게 GUI 화면을 구성할 수 있는 툴(또는 프로그램)이 있으면 좋겠다는 생각을 많이들 할 겁니다. 이번 절에서 배울 Qt Designer 가 바로 위지위그 방식으로 UI 구성을 도와주는 프로그램입니다. 여러분도 이 툴을 사용하면 귀찮은 GUI 작업에서 해방될 수 있습니다.

그림 16.9 는 Qt Designer 를 실행한 모습입니다. 왼쪽에 있는 위젯을 드래그 앤드 드롭해서 쉽게 레이아웃을 구성할 수 있습니다.

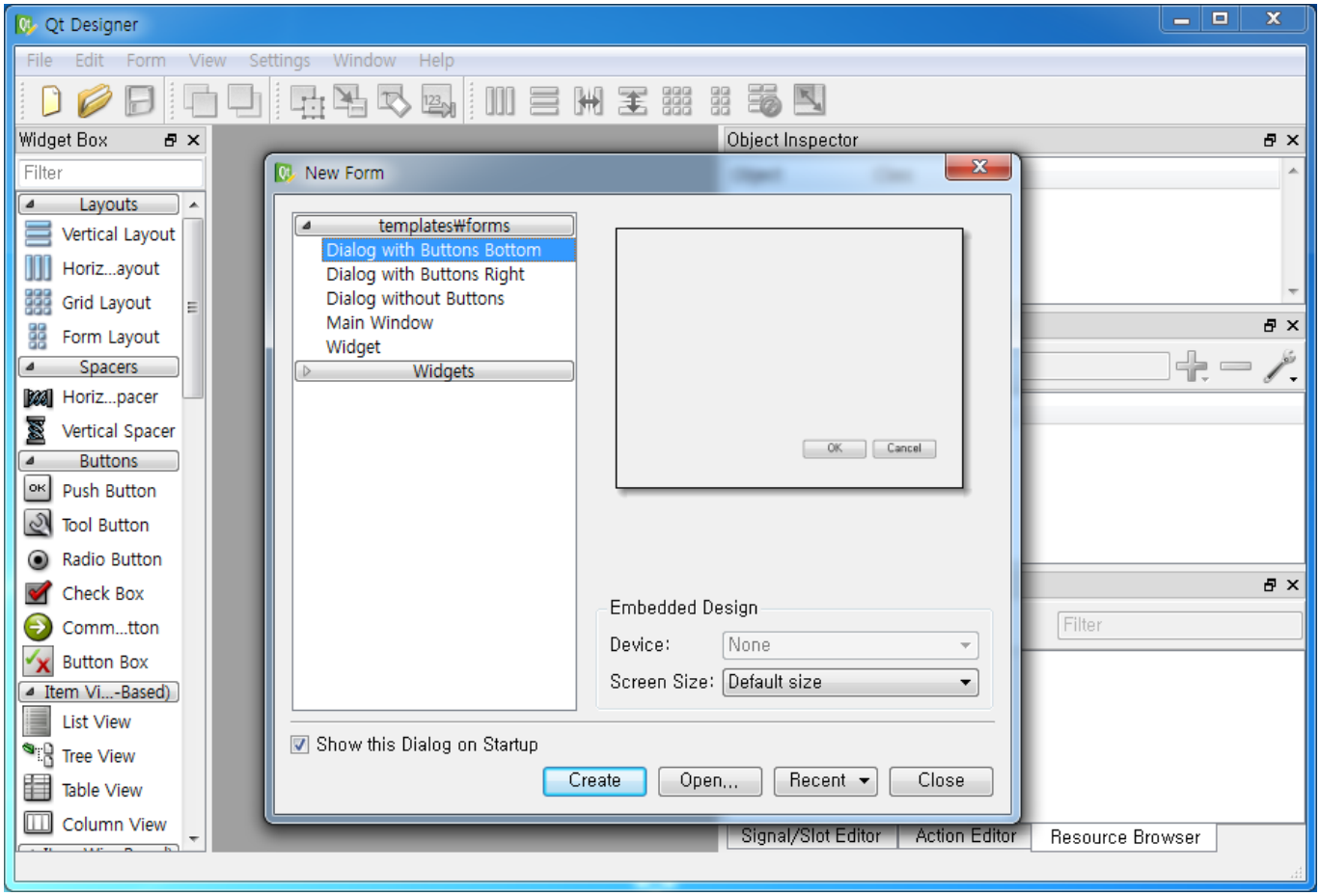


그림 16.9 Qt Designer

2) Qt Designer 를 이용한 UI 구성

이번 절에서는 Qt Designer 를 이용해 아주 간단한 형태의 UI 를 구성해 보겠습니다. Qt Designer 는 하나의 독립적인 프로그램으로 아나콘다 배포판에 포함돼 있습니다. 보통은 PyQt 를 설치할 때 함께 설치됩니다.

그림 16.10 과 같이 아나콘다 배포판에는 C:\Anaconda3\Library\bin 디렉터리에 designer.exe 파일이 존재합니다. 이 파일이 바로 Qt Designer 실행 파일입니다. 윈도우에서는 해당 파일을 더블클릭하면 Qt Designer 를 실행할 수 있습니다.

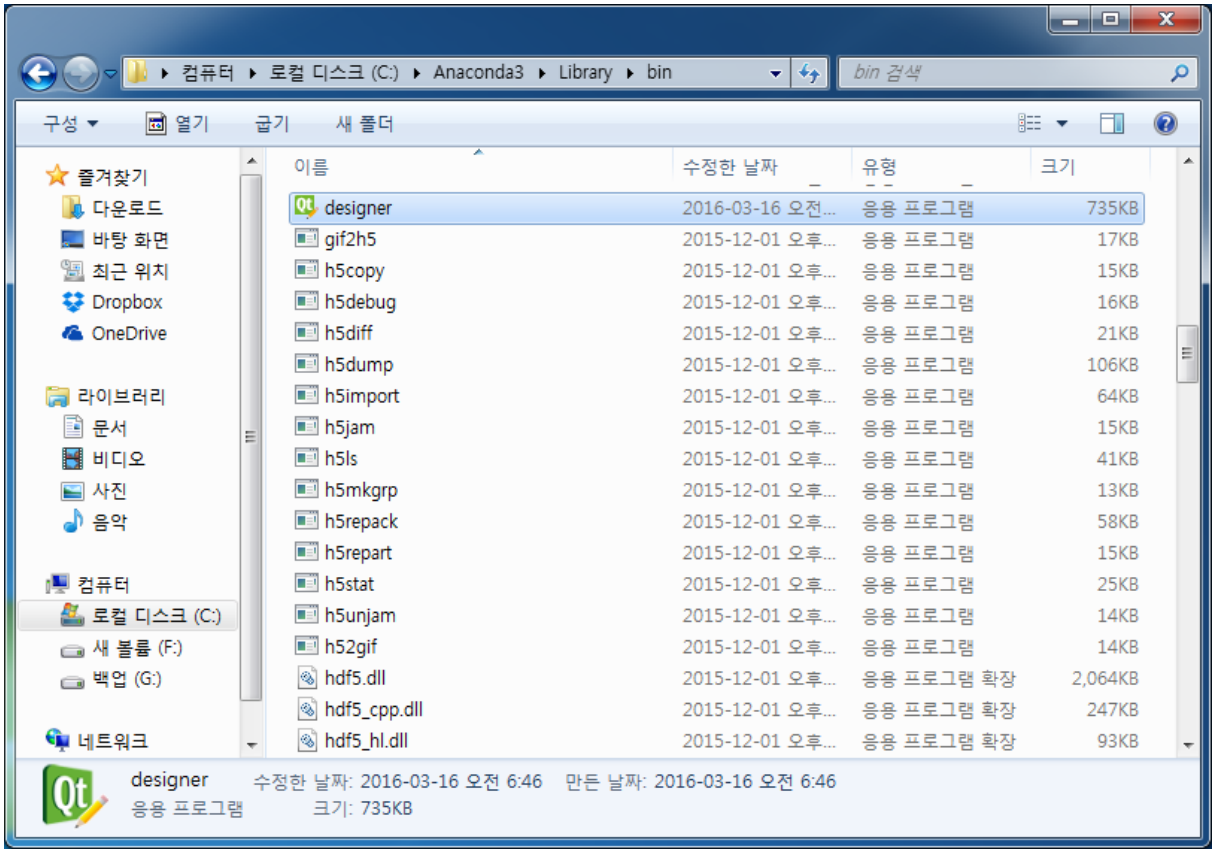


그림 16.10 Qt Designer 실행 파일 경로

Qt Designer 를 실행했다면 가장 먼저 할 일은 Dialog 나 Main Window 를 생성하는 일입니다. 그림 16.11 과 같이 [Main Window]를 선택한 후 [Create] 버튼을 눌러 윈도우를 생성합니다.

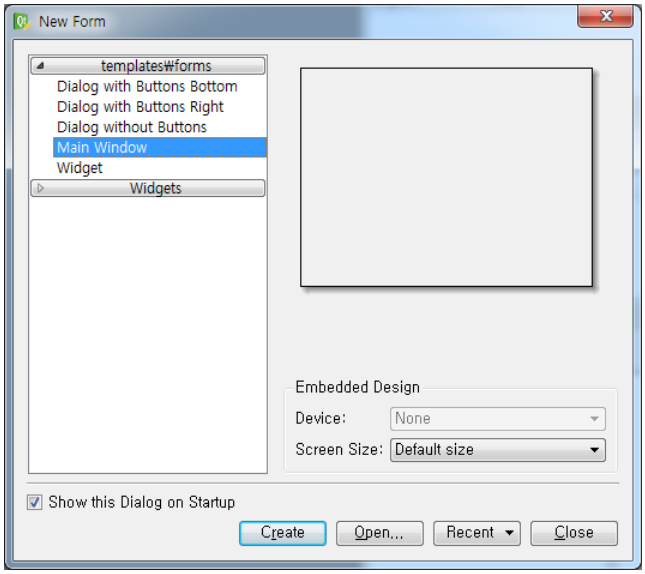


그림 16.11 Main Window 생성

윈도우를 생성하면 그림 16.12 와 같이 Qt Designer 의 중앙에 윈도우가 나타납니다. 윈도우의 오른쪽 아래를 클릭해 윈도우 크기를 직접 조절할 수도 있습니다. 이때 한 줄의 코드도 작성하지 않았지만 QMainWindow 클래스를 상속받아 MyWindow 클래스를 정의하고 해당 클래스에 대한 객체를 생성한 것과 비슷한 윈도우가 나타났습니다.

그림 16.12 를 보면 생성된 윈도우의 왼쪽에는 위젯 Box 가 보이고, 오른쪽에는 Object Inspector 나 Property Editor 가 보입니다. 앞으로 여러분은 왼쪽에 있는 위젯들을 윈도우로 드래그 앤드 드롭해서 위치시킨 후 Object Inspector 나 Property Editor 를 통해 값을 변경하게 될 것입니다.

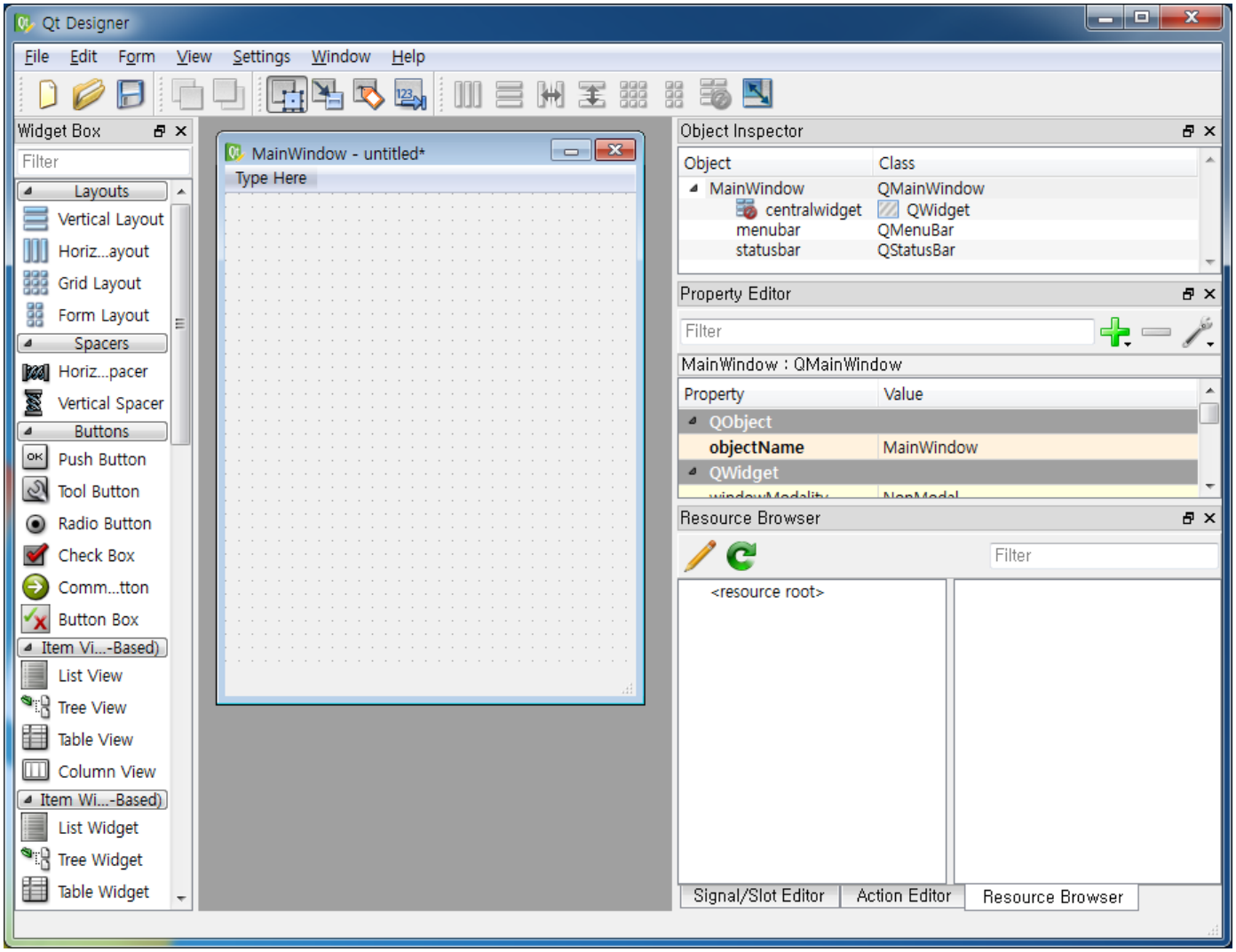


그림 16.12 Qt Designer 구성

아직 감흥이 없는 분들을 위해 위젯 Box 에서 Push Button 위젯과 Label 위젯을 선택한 후 윈도우로 드래그 앤드 드롭해서 그림 16.13 과 같이 위젯들을 배치해 보겠습니다.

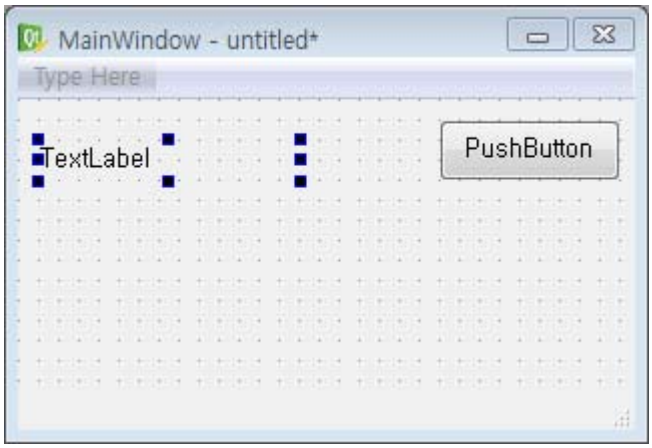


그림 16.13 Qt Designer 를 사용한 레이아웃 디자인

위젯이 배치됐다면 이번에는 Property Editor 를 이용해 각 위젯의 속성값을 변경해 보겠습니다. 그림 16.14 와 같이 MainWindow 의 PushButton 을 클릭한 상태에서 Property Editor 의 text 항목의 속성값을 '클릭'으로 변경합니다. Property Editor 에서 속성값을 변경하면 바로 MainWindow 에 반영되는 것을 확인할 수 있습니다.

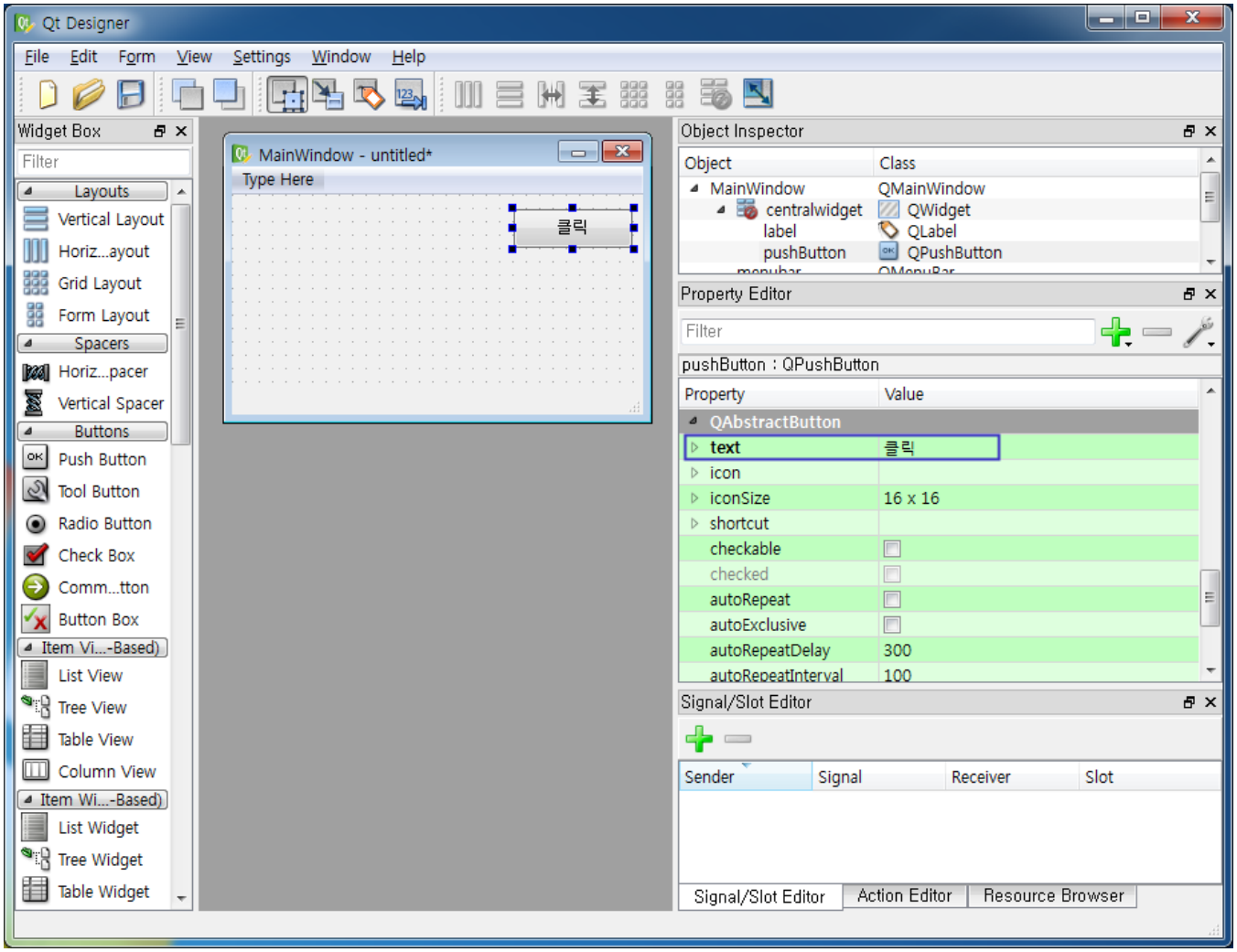


그림 16.14 Property Editor 를 이용한 속성 변경-1

이번에는 그림 16.15 와 같이 MainWindow 에서 Label 을 선택한 후 Property Editor 의 text 속성값을 공백 상태로 비워 둡니다. 공백 상태로 두면 Label 객체에 아무것도 출력되지 않습니다.

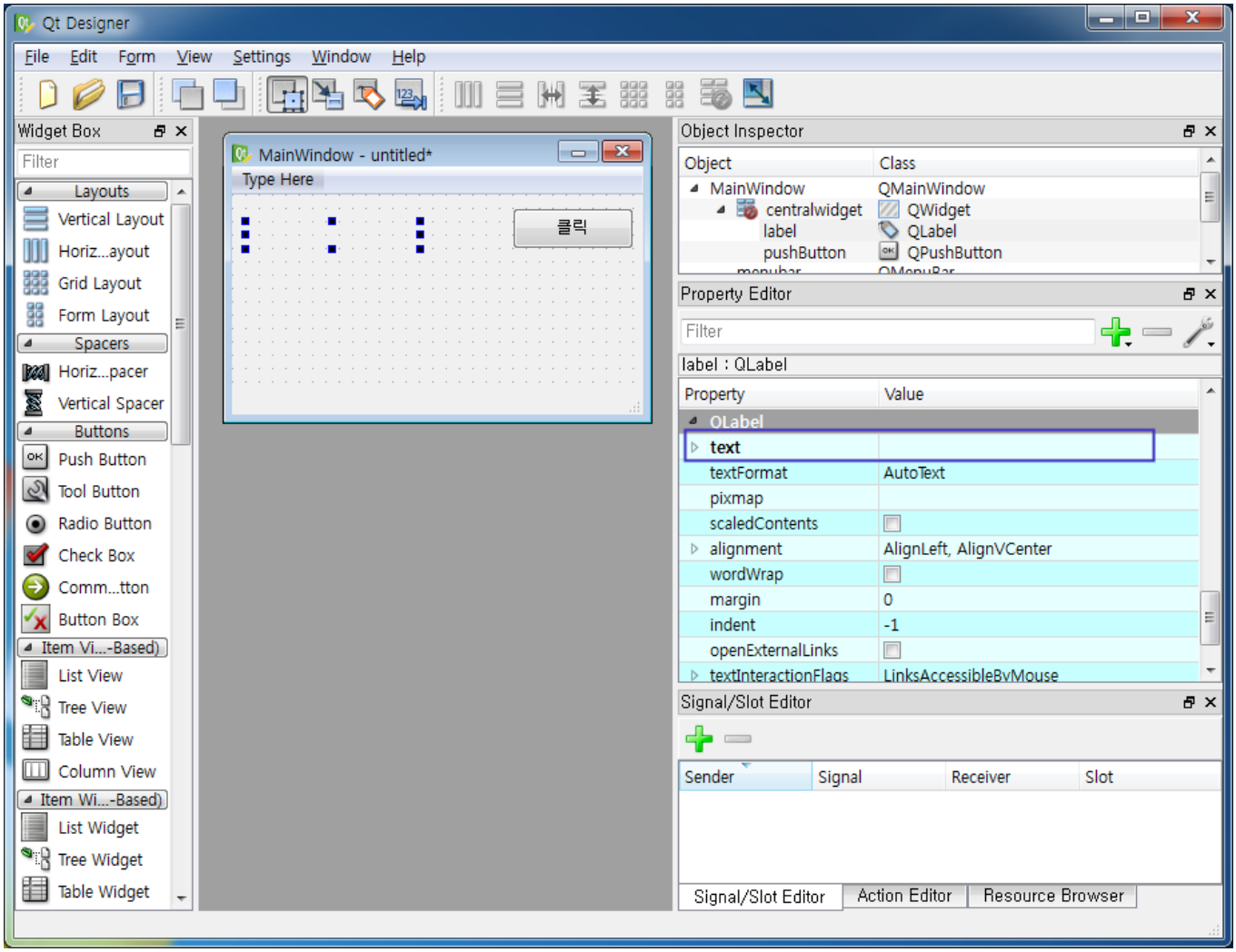


그림 16.15 Property Editor 를 이용한 속성 변경-2

지금까지 작업한 내용에 대한 결과물을 미리 보려면 단축키인 Ctrl + R 을 누르면 됩니다. Ctrl + R 을 누르면 그림 16.16 과 같이 작업 중인 UI 의 최종 결과물인 윈도우가 출력되는 것을 확인할 수 있습니다. 다만 아직 시그널과 슬롯은 따로 처리하지 않았기 때문에 버튼을 클릭해도 어떤 동작도 수행되지 않습니다.

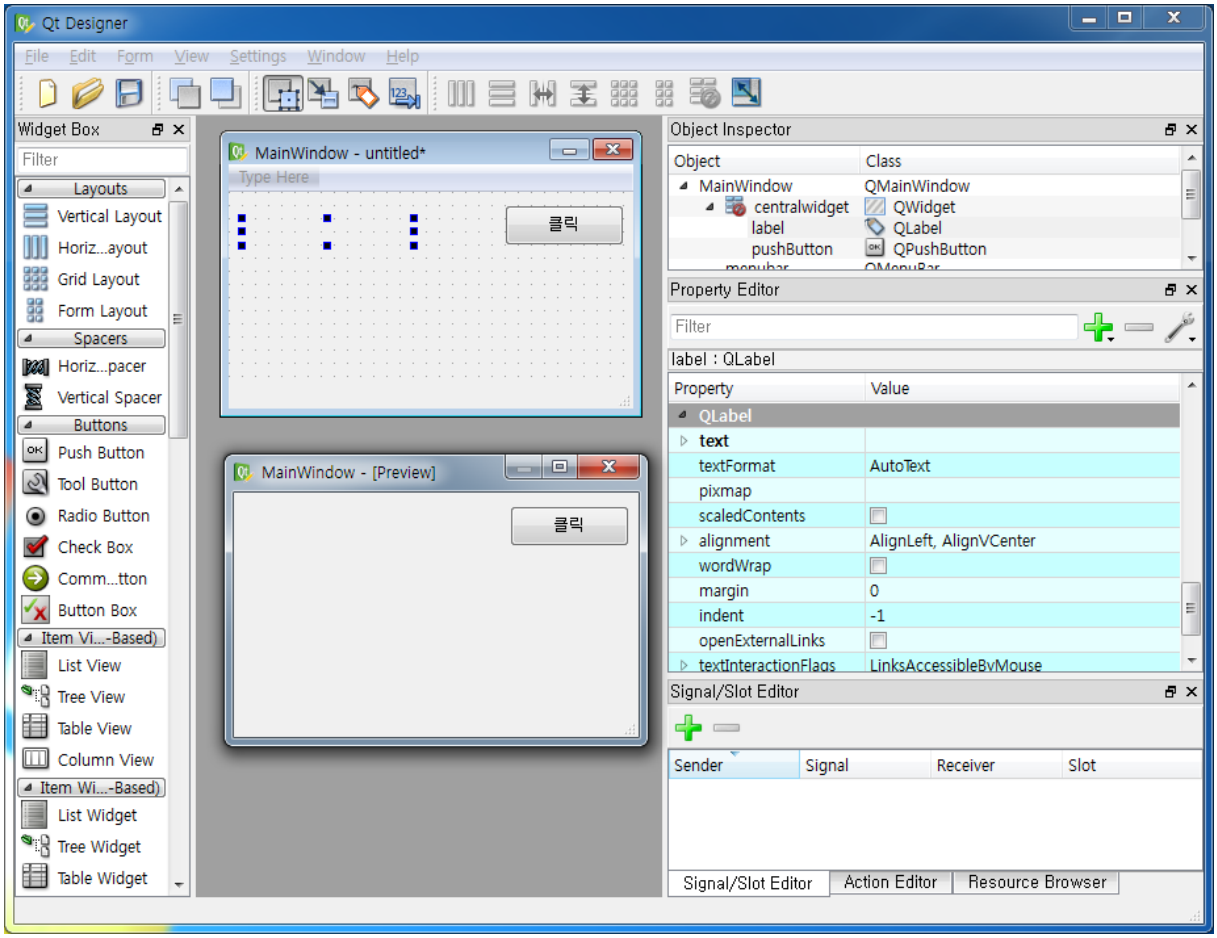


그림 16.16 미리 보기(Ctrl+R 단축키)

구성한 UI 가 마음에 든다면 이를 저장합니다. Qt Designer 의 툴바에서 디스켓 모양의 아이콘을 클릭하거나 [File] → [Save] 메뉴를 차례로 클릭하면 'Save Form As' 윈도우가 나타납니다. 그림 16.17 과 같이 윈도우 바탕 화면에 'main_window'라는 이름으로 저장합니다.

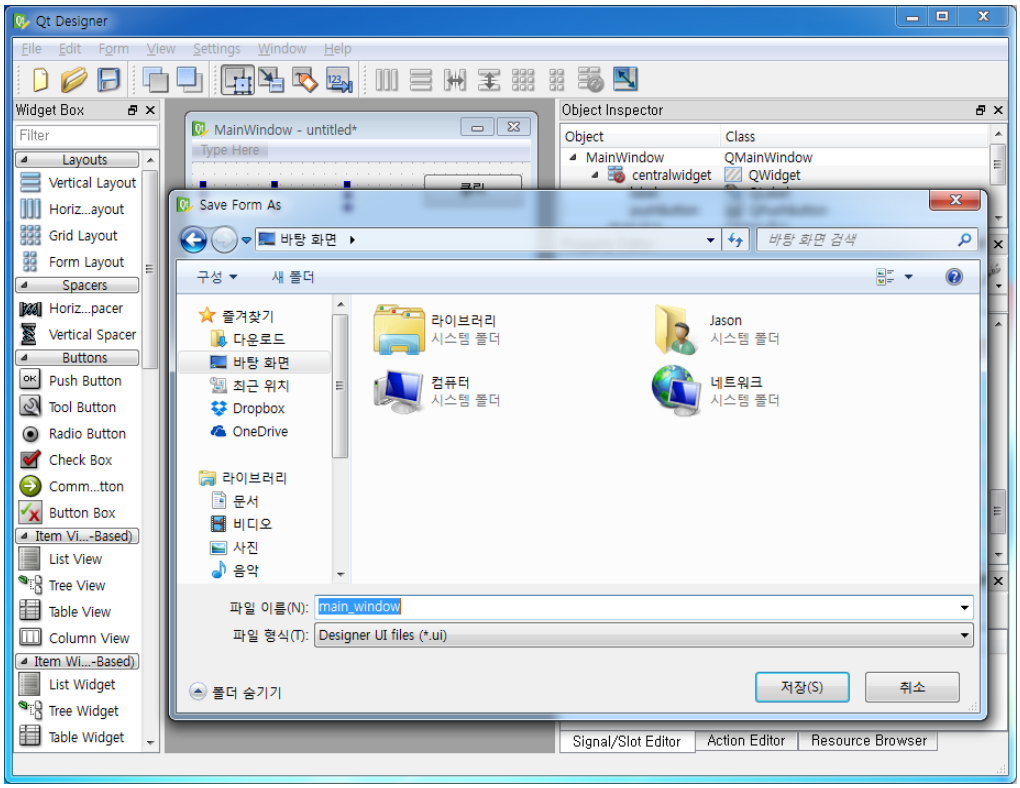


그림 16.17 UI 파일 저장

윈도우 메모장을 실행한 후 바탕 화면에 저장된 main_window.ui 파일을 열어보면 지금까지 구성한 UI 레이아웃이 XML 이라는 형식으로 저장된 것을 확인할 수 있습니다.

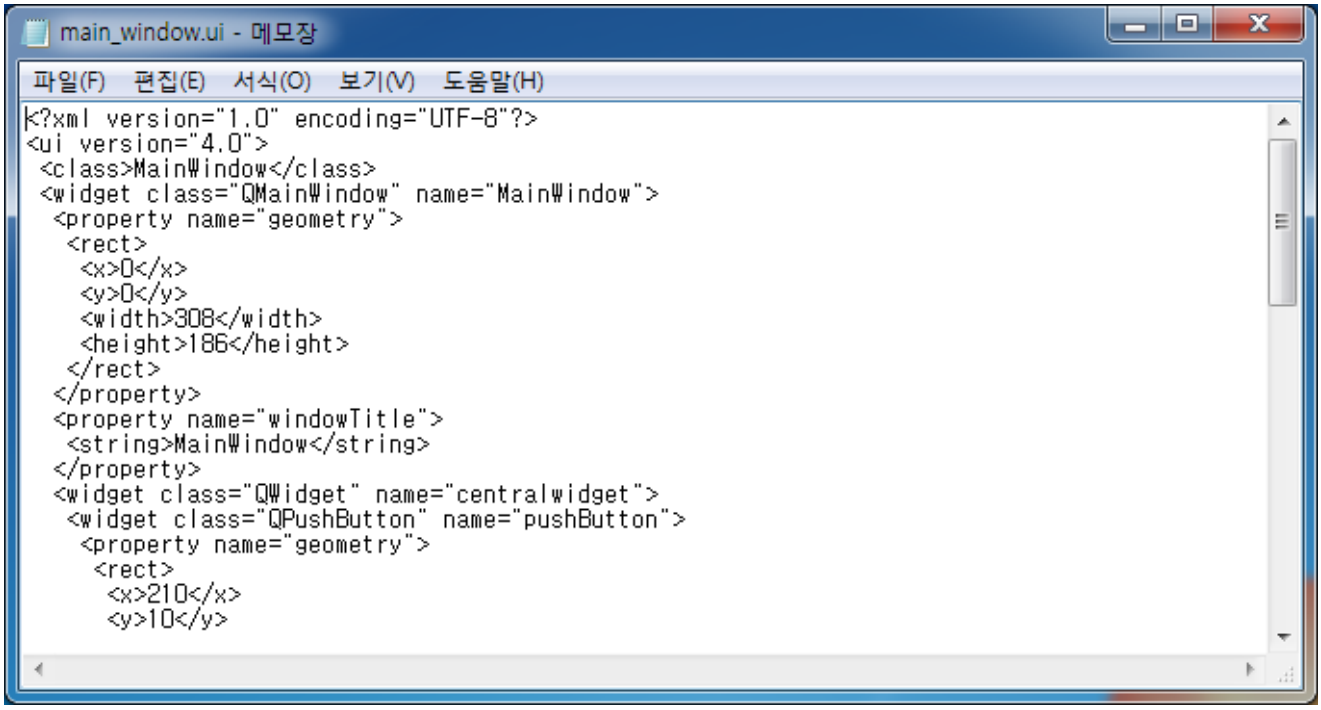


그림 16.18 Qt Designer UI 파일의 내용(XML 형식)

3) UI 파일을 파이썬 코드로 변환하기

앞 절에서 Qt Designer 를 통해 main_window.ui 라는 파일을 만들었습니다. 생성된 UI 파일을 사용해 GUI 를 실행하는 방법에는 두 가지가 있습니다. 이번 절에서는 그중 첫 번째 방법에 대해 알아보겠습니다.

앞서 설명한 것처럼 Qt Designer 의 결과 파일인 UI 파일은 XML 이라는 언어로 구성돼 있습니다. 파이썬은 XML 코드를 바로 실행할 수 없으므로 XML 코드를 파이썬 코드로 변환하는 것이 필요합니다.

PyQt5 는 XML 로 구성된 UI 파일을 파이썬 코드로 자동으로 변환하는 pyuic.py 라는 프로그램을 제공합니다. 확장자를 보면 알겠지만 이 프로그램 역시 파이썬으로 구현돼 있습니다. pyuic.py 파일은 XML 파일을 입력받은 후 이를 파이썬 코드로 변환합니다.

먼저 윈도우 바탕 화면에 저장된 main_window.ui 라는 파일을 C:\Anaconda3\Lib\site-packages\PyQt5\uic 라는 디렉터리로 복사합니다.

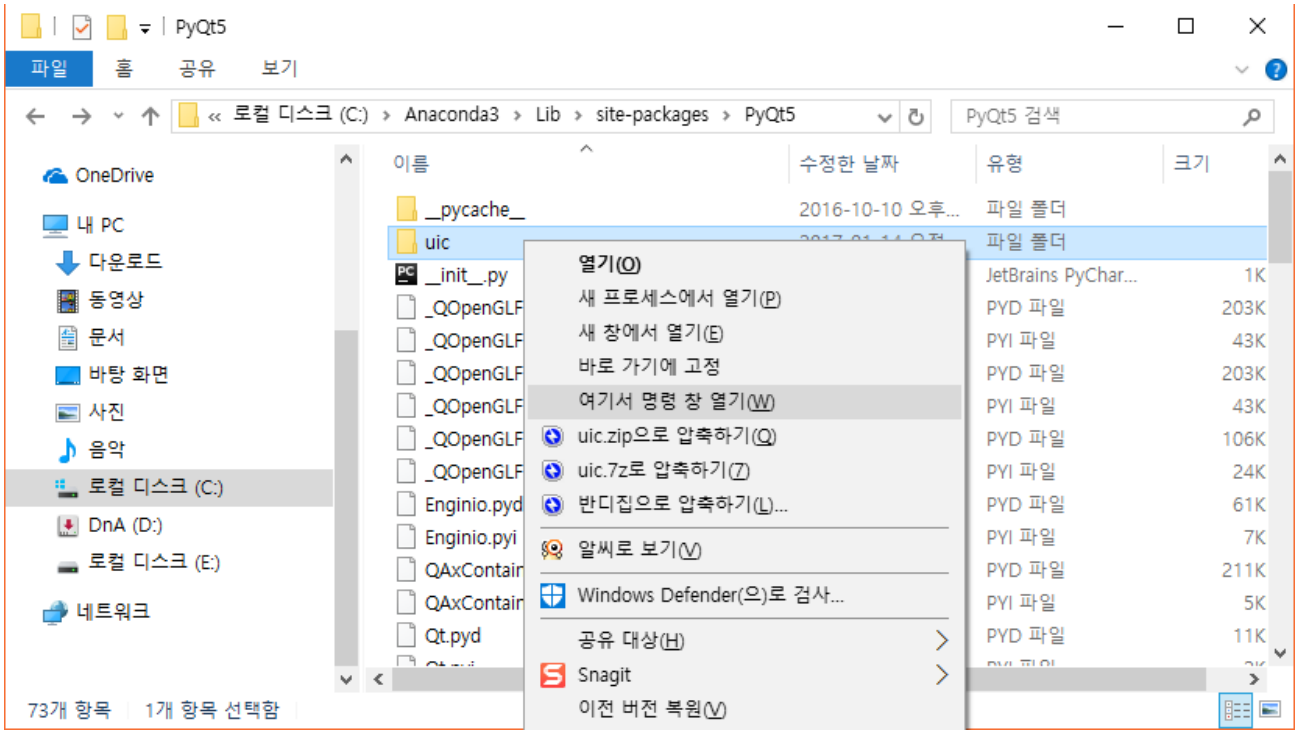


그림 16.19 uic 디렉터리에서 명령 창 열기

그림 16.19 와 같이 C:\Anaconda3\Lib\site-packages\PyQt5 디렉터리에서 uic 디렉터리를 선택한 후 Shift 키를 누른 상태에서 마우스 오른쪽 버튼을 클릭합니다. 그림 16.19 와 같은 메뉴가 나타나면 [여기서 명령 창 열기] 메뉴를 선택합니다.

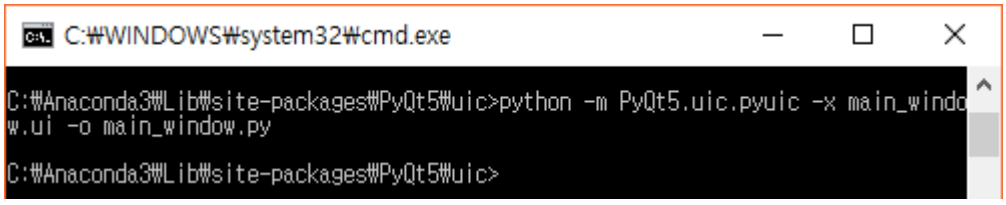


그림 16.20 pyuic 를 이용해 UI 파일을 파이썬 코드로 변환

명령 창이 실행되면 그림 16.20 과 같이 명령어를 입력해 main_window.ui 파일을 main_window.py 파일로 변환합니다. 에러 없이 정상적으로 변환이 수행되면 C:\Anaconda3\Lib\site-packages\PyQt5\uic 디렉터리에 main_window.py 파일이 생성됩니다. 생성된 main_window.py 파일을 열어보면 그림 16.21 과 같이 파이썬 코드로 구성된 것을 확인할 수 있습니다.

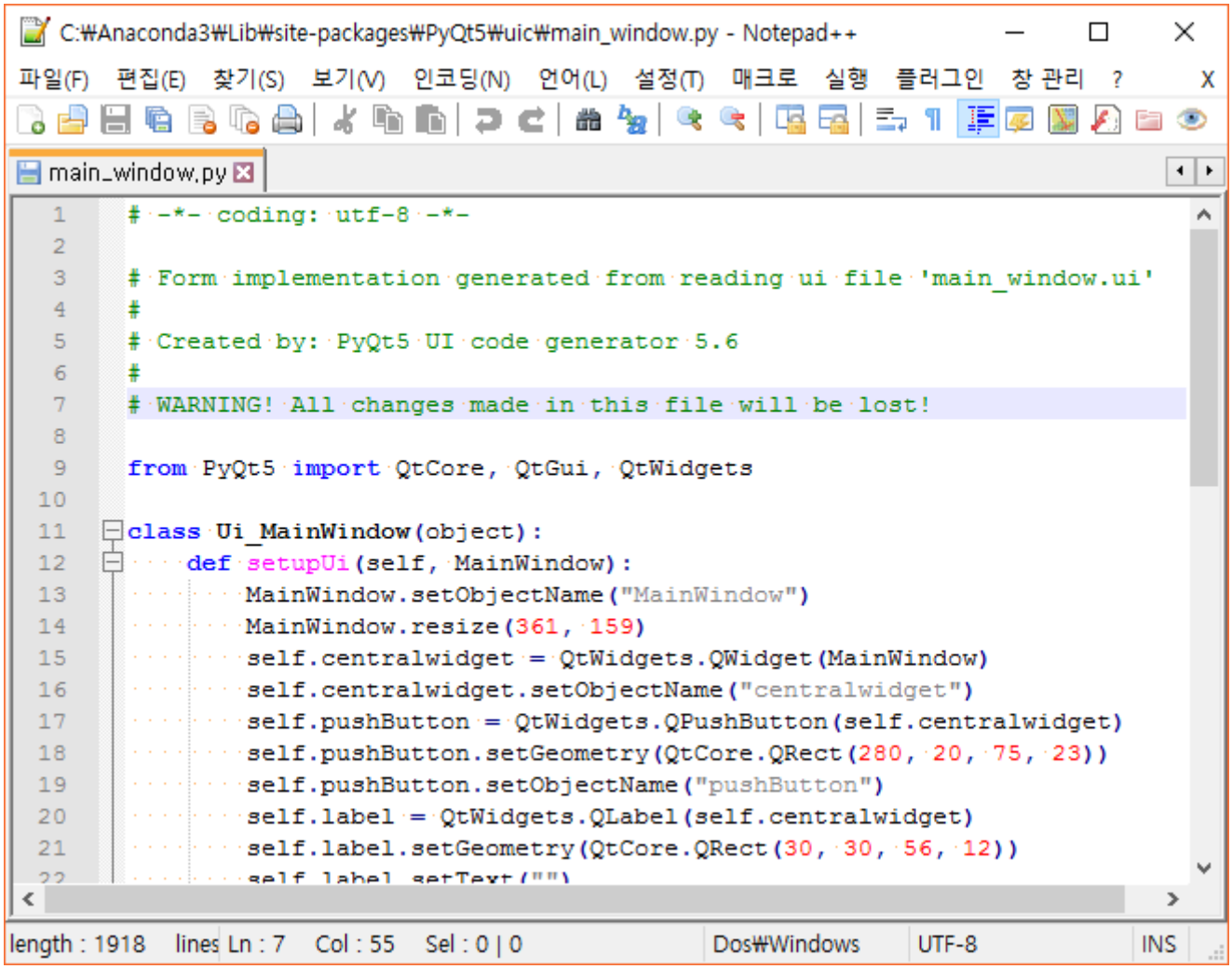


그림 16.21 UI 파일로부터 생성된 파이썬 코드

UI 파일로부터 생성된 파이썬 코드(main_window.py)를 실행하기 위해 그림 16.22 와 같이 명령 창에서 명령을 수행합니다. 그림 16.22 를 살펴보면 앞서 Qt Designer 를 통해 구성했던 UI 가 그대로 화면에 출력됨을 확인할 수 있습니다.

이처럼 Qt Designer 를 이용하면 UI 구성을 위한 코드를 작성하지 않고도 편리하게 UI 를 구성하고 프로그램을 동작시킬 수 있습니다. 물론 아직은 버튼에 대한 이벤트(signal)를 처리하는 메서드 (slot)를 구현하지 않았기 때문에 버튼을 클릭해도 아무런 동작을 수행하지 않습니다.

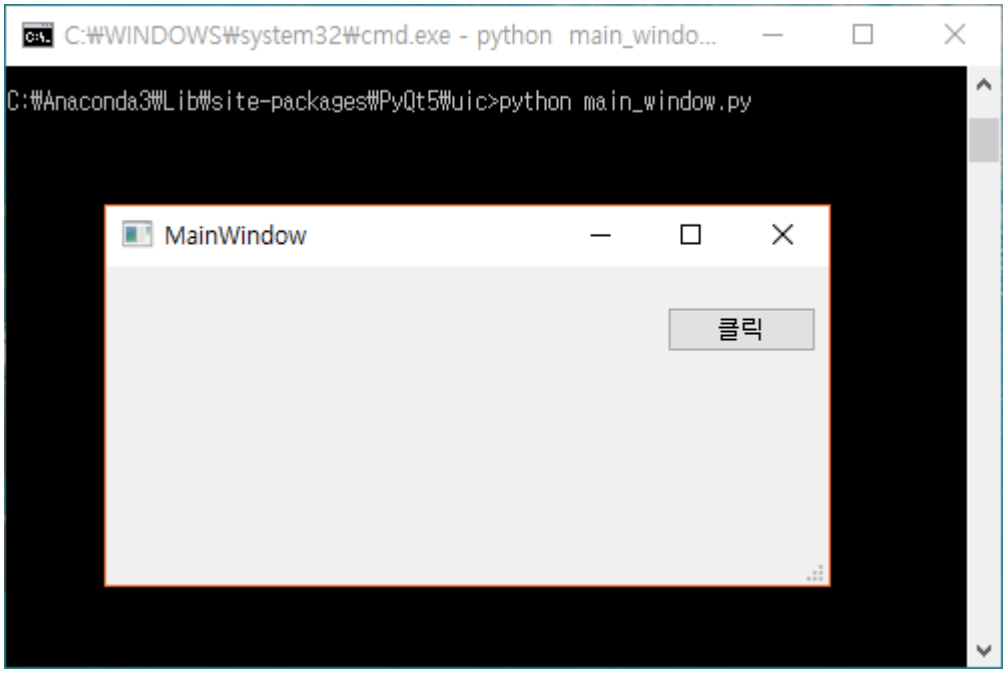


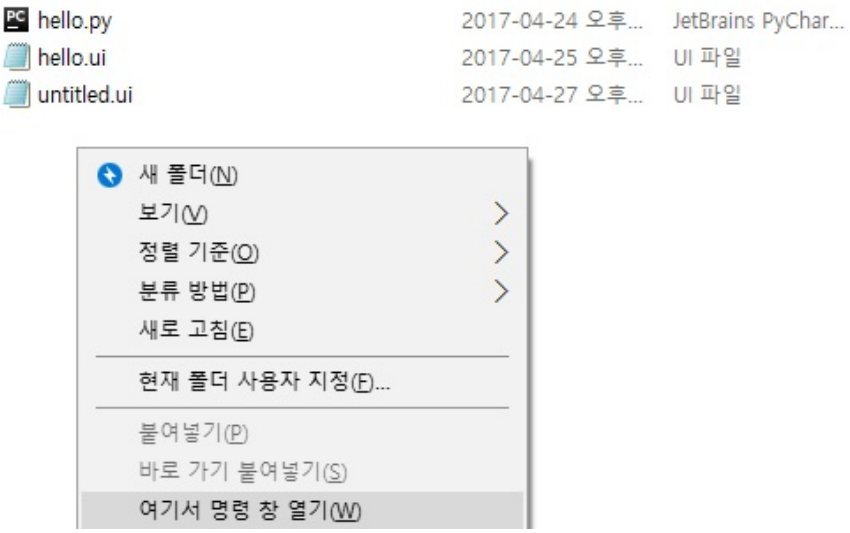
그림 16.22 UI 파일로부터 생성된 파이썬 코드를 실행

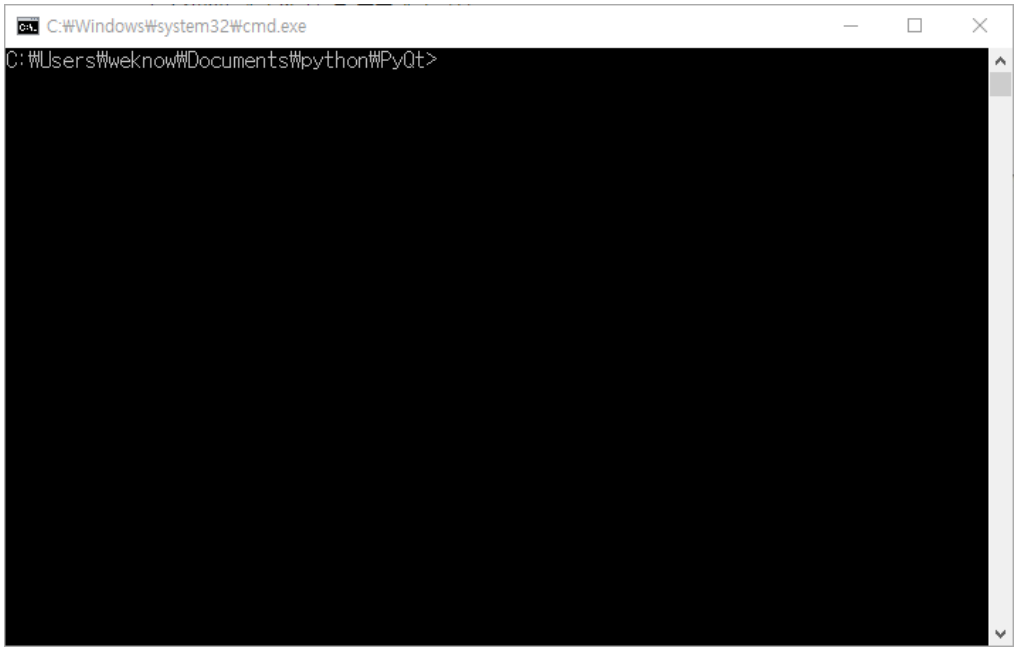
==========

PyQt designer 로 만든 *.ui 파일을 사용하는 방법은 두가지가 있습니다.
1. Python 코드로 변환 후 사용
2. Python 에서 ui 파일을 로드해서 사용.

1 번의 방법을 알아봅시다.

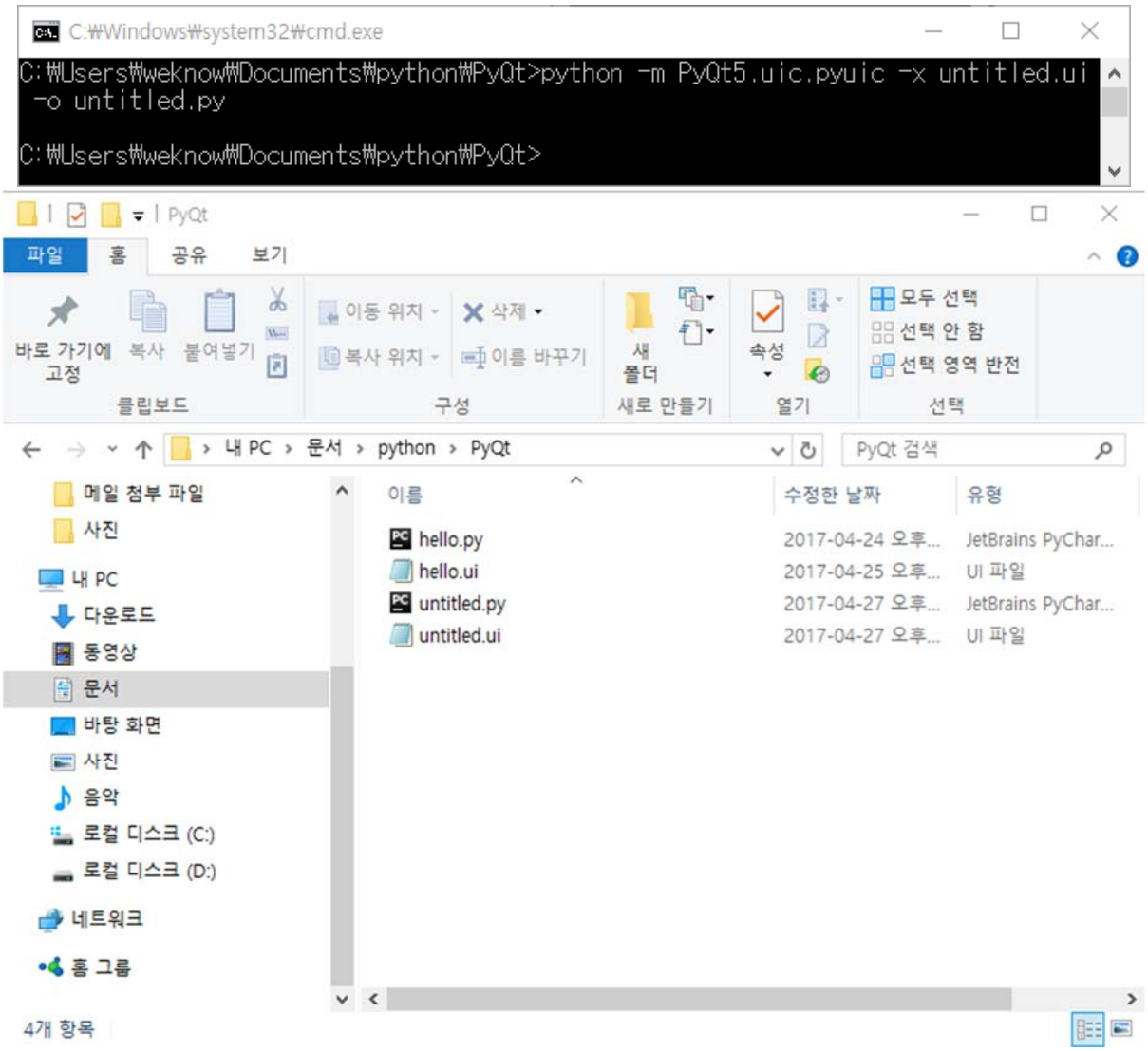
우선 만들어 놓은 ui 가 있는 폴더로 이동합니다.
untitled.ui 파일을 untitled.py 파일로 변환해보겠습니다.
'shift + 우클릭 > 여기서 명령 창 열기' 를 선택합니다.





명령 프롬프트가 실행이되면 명령어를 통해 ui 파일을 변환시켜 줍니다.

```
python -m PyQt5.uic.pyuic -x untitled.ui -o untitled.py
```



untitled.py 파일이 생성된 것을 확인할 수 있습니다.
내용을 확인해 봅시다.

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'untitled.ui'
#
# Created by: PyQt5 UI code generator 5.6
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(291, 149)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(50, 30, 56, 12))
        self.label.setObjectName("label")
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(150, 20, 75, 23))
        self.pushButton.setObjectName("pushButton")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 291, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.label.setText(_translate("MainWindow", "TextLabel"))
        self.pushButton.setText(_translate("MainWindow", "PushButton"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
```

untitled.py

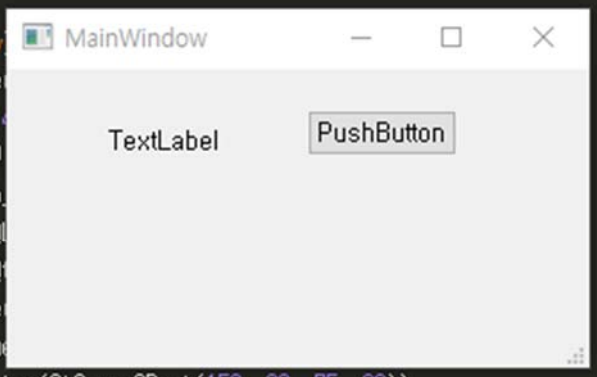
저는 pycharm 을 사용하므로 바로 Ctrl+F9 로 실행해보겠습니다.

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'untitled.ui'
#
# Created by: PyQt5 UI code generator 5.6
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def setupUi(self, MainWindow
        MainWindow.setObjectName
        MainWindow.resize(291, 1
        self.centralwidget = QtW
        self.centralwidget.setOb
        self.label = QtWidgets.Q
        self.label.setGeometry(Q
        self.label.setObjectName
        self.pushButton = QtWidg
        self.pushButton.setGeometry(QtCore.QRect(150, 20, 75, 23))
        self.pushButton.setObjectName("pushButton")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 291, 21))
```



untitled.py 실행화면

4) UI 파일을 파이썬 코드에서 로드하기

앞 절에서는 UI 파일을 pyuic.py 파일을 사용해 파이썬 코드로 변환한 후 이를 실행하는 방법을 알아봤습니다. 이 같은 방법은 편리하긴 하지만 UI 가 변경될 때마다 UI 파일을 파이썬으로 변경해야 하는 문제가 있습니다. 만약 UI 파일로부터 변환된 파이썬 코드에 이벤트 처리를 위한 코드를 함께 구현했다면 문제는 더 심각해질 것입니다.

이번 절에서는 UI 파일을 파이썬 코드로 변환하지 않고 사용하는 방법을 알아보겠습니다. 앞 절에서 main_window.ui 파일을 윈도우 바탕 화면에 저장했습니다. 예제 16.2 의 코드를 run.py 라는 파일로 저장한 후 main_window.ui 파일이 있는 경로인 윈도우 바탕 화면에 저장합니다.

```
import sys
from PyQt5.QtWidgets import *
from PyQt5 import uic

form_class = uic.loadUiType("main_window.ui")[0]

class MyWindow(QMainWindow, form_class):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    myWindow = MyWindow()
    myWindow.show()
    app.exec_()
```

예제 16.2 ui 파일을 사용한 UI 구성(소스코드: book/ch16/04.py)

예제 16.2 를 실행하면 그림 16.23 과 같은 윈도우가 화면에 출력됩니다. 이는 앞서 UI 파일을 파이썬 코드로 변환한 후 변환된 파이썬 코드를 실행했을 때와 같습니다.

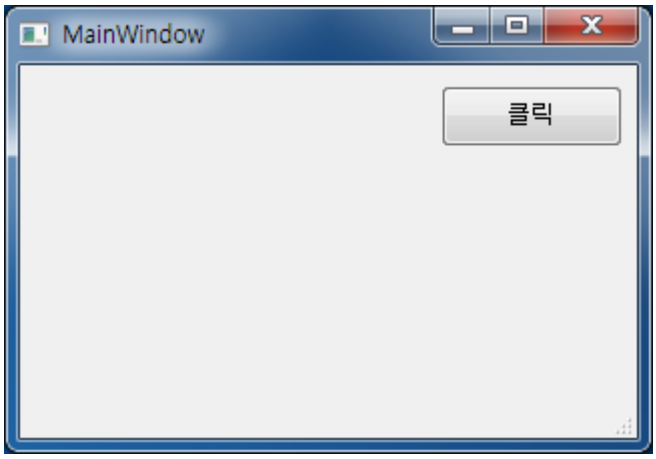


그림 16.23 UI 파일 로드를 통한 GUI 프로그램 실행

예제 16.2 를 살펴보면 QApplication 클래스를 이용해 이벤트 루프를 생성하는 부분과 MyWindow 라는 클래스를 정의한 부분이 있습니다. MyWindow 클래스를 살펴보면 QMainWindow 와 form_class 클래스로부터 다중 상속을 받았음을 확인할 수 있습니다. 여기서 form_class 는 main_window.ui 파일을 로드해서 만든 클래스입니다.

MyWindow 클래스는 form_class 를 상속받았기 때문에 form_class 에 정의돼 있던 속성이나 메서드를 모두 상속받게 됩니다. 따라서 다음과 같이 setupUi 메서드를 호출함으로써 Qt Designer 를 통해 구성한 UI 를 화면에 출력할 수 있습니다. 참고로 setupUi 라는 메서드 이름은 정해진 이름이기 때문에 그대로 사용해야 합니다.

```
self.setupUi(self)
```

이번 절에서 살펴본 방법은 UI 파일을 파이썬 코드로 변환하지 않고 파이썬 코드에서 UI 파일을 로드했습니다. 이 방식은 추후에도 UI 파일을 Qt Designer 를 통해 쉽게 수정할 수 있으며, UI 파일이 변경되더라도 파이썬 코드의 변경을 최소화할 수 있습니다.

5) 이벤트 처리하기

Qt Designer 를 사용하면 더 쉽고 편리하게 UI 를 구성할 수 있음을 배웠습니다. 하지만 사용자 이벤트를 처리하는 메서드나 함수는 여전히 여러분이 직접 구현해야 합니다. 이번 절에서는 그림 16.23 의 윈 도우에서 버튼을 클릭할 때 발생하는 이벤트를 처리하는 프로그램을 작성해 보겠습니다.

예제 16.3 은 예제 16.2 의 기본 윈도우 구성 코드에 시그널(self.pushButton.clicked)과 슬롯(btn_clicked)을 연결하는 코드와 슬롯 구현부를 추가한 것입니다. 여기서 중요한 점은 여전히 UI 의 구성은 main_window.ui 파일을 사용한다는 점입니다.

```
import sys
from PyQt5.QtWidgets import *
from PyQt5 import uic

form_class = uic.loadUiType("main_window.ui")[0]

class MyWindow(QMainWindow, form_class):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.pushButton.clicked.connect(self.btn_clicked)

    def btn_clicked(self):
        QMessageBox.about(self, "message", "clicked")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    myWindow = MyWindow()
    myWindow.show()
    app.exec_()
```

예제 16.3 슬롯 구현 및 시그널-슬롯 연결(소스코드: book/ch16/05.py)

예제 16.3 을 살펴보면 버튼 객체에 접근하기 위해 self.pushButton 을 사용했습니다. 그런데 self.pushButton 이라는 변수는 예제 16.3 에서 어떤 객체도 바인딩하고 있지 않습니다. 그 이유는 버튼 객체에 대한 생성 및 바인딩은 setupUi 메서드에서 수행되기 때문입니다.

그렇다면 우리는 어떻게 객체를 바인딩하고 있는 변수를 알 수 있을까요? 그림 16.24 와 같이 Qt Designer 를 실행한 후 main_window.ui 파일을 엽니다. 버튼을 클릭한 상태로 Property Editor 를 살펴보면 객체의 이름이 pushButton 임을 확인할 수 있습니다. 물론 이 변수는 여러분이 언제든지 변경할 수있지만중요한것은이값이파이썬코드에서해당객체에접근하는데사용된다는점입니다.

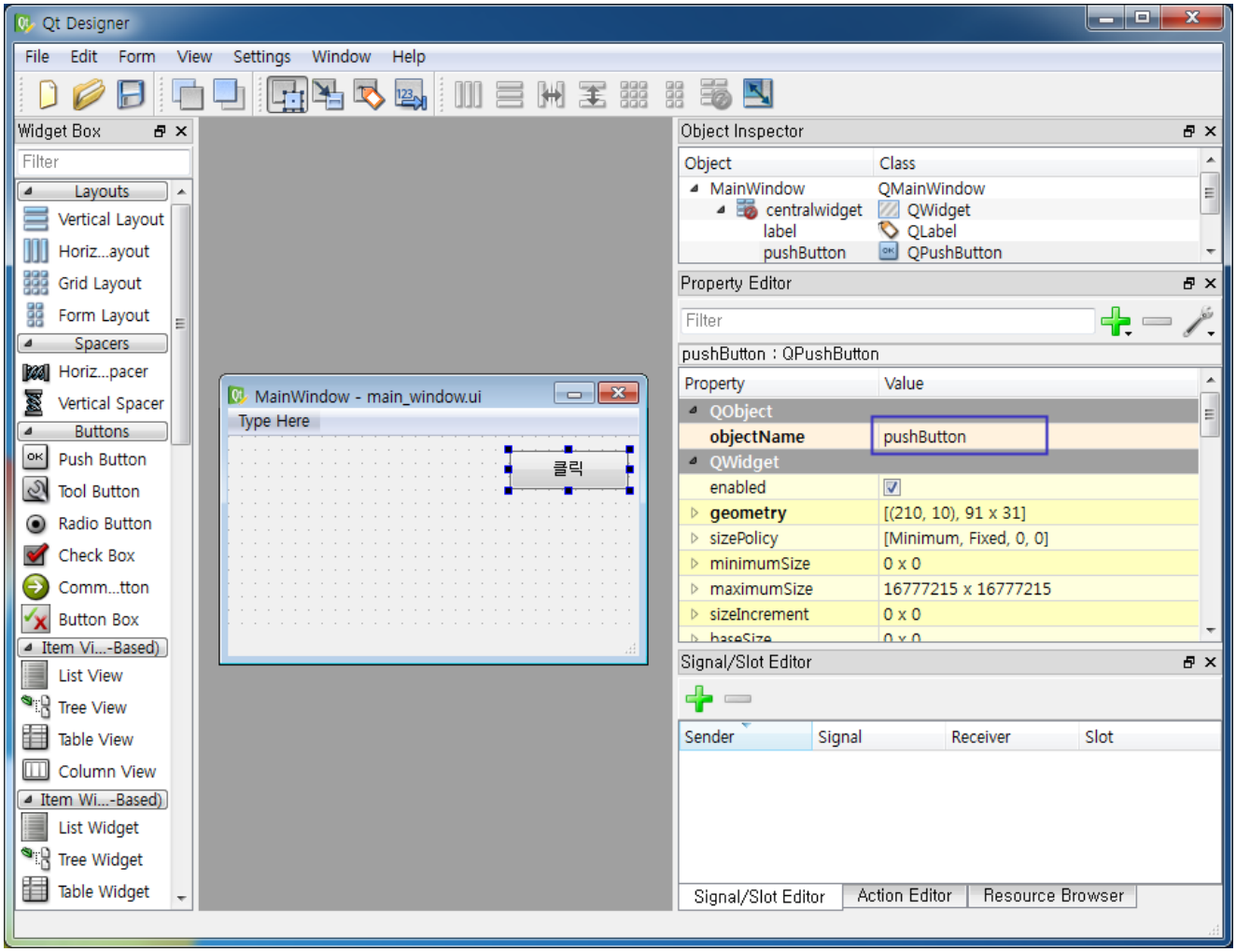


그림 16.24 버튼 객체의 QObject 값 확인

예제 16.3 을 살펴보면 self.pushButton 이라는 변수가 바인딩하고 있는 객체인 버튼에서 'clicked'라는 시그널이 발생하면 self.btn_clicked 라는 메서드가 호출되는 것을 확인할 수 있습니다. btn_clicked 메서드는 QMessageBox 를 통해 화면에 메시지 박스를 출력하도록 구현돼 있습니다.

예제 16.3 을 실행한 후 버튼을 클릭하면 그림 16.25 와 같이 메시지 박스가 화면에 출력되는 것을 확인할 수 있습니다.

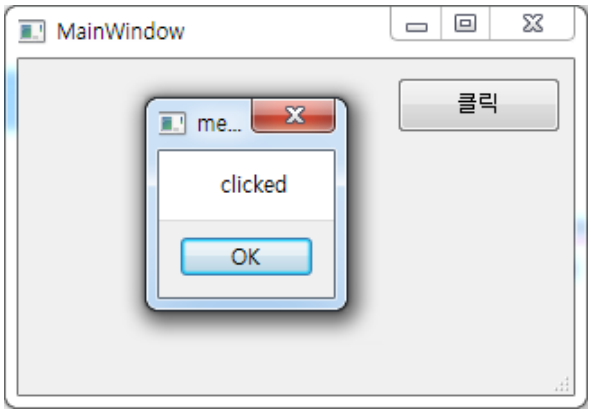


그림 16.25 슬롯을 통한 시그널 처리

https://wikidocs.net/5220

3. 기본 위젯

PyQt 에서 버튼, 메뉴 등과 같이 UI 를 구성하는 데 사용되는 요소를 위젯이라고 합니다. GUI 프로그래밍은 여러 위젯을 활용해 최적의 UI 를 구성한 후 각 위젯에서 발생하는 이벤트를 적절히 처리하는 것입니다. 이번 절에서는 PyQt 에서 자주 사용하는 위젯을 살펴보겠습니다.

1) QPushButton

QPushButton 은 버튼을 생성하기 위해 사용하는 위젯입니다. UI 를 구성할 때 버튼은 가장 기본이 되는 위젯으로서 사용자로부터 '예', '아니오' 같은 이벤트를 받는 데 사용합니다. 그림 16.26 은 윈도우에 QPushButton 위젯으로 생성한 버튼이 있습니다. 사용자가 '닫기'라는 버튼을 클릭하면 윈도우가 종료됩니다.

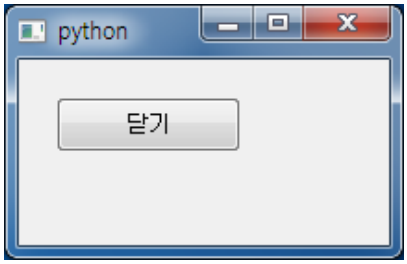


그림 16.26 QPushButton 위젯

예제 16.4 QPushButton 사용 예(소스코드: book/ch16/06.py)

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        btn1 = QPushButton("닫기", self)
        btn1.move(20, 20)
        btn1.clicked.connect(QCoreApplication.instance().quit)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

예제 16.4 는 그림 16.26 에 나온 프로그램의 소스코드입니다. 전체적인 코드 구성은 예제 16.1 과 비슷합니다. 단지 QPushButton 객체에서 'clicked'라는 시그널이 발생하면 이를 처리하는 슬롯으로 QCoreApplication.instance().quit 과 연결한 부분이 다릅니다.

참고로 QCoreApplication.instance()를 이용하면 예제 16.4 에서 app 이라는 변수가 바인딩하고 있는 동일한 객체를 얻어올 수 있습니다. app 가 바인딩하고 있는 객체는 QApplication 클래스의 인스턴스인데 해당 객체는 quit 이라는 메서드를 포함합니다. 버튼을 클릭했을 때 QApplication 이 제공하는 quit 메서드를 호출하도록 연결함으로써 윈도우가 종료하는 것입니다.

예제 16.5 QPushButton 사용 예-2(소스코드: book/ch16/07.py)

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

app = None

class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        btn1 = QPushButton("닫기", self)
        btn1.move(20, 20)
        btn1.clicked.connect(app.quit)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

예제 16.4 에서 QCoreApplication.instance()가 잘 이해되지 않는다면 예제 16.5 를 살펴보기 바랍니다. 예제 16.5 는 예제 16.4 와 UI 를 구성하는 코드는 같지만 app 라는 변수를 전역적으로 선언함으로써 클래스 내에서도 app 이라는 변수를 사용할 수 있게 했습니다.

예제 16.5 를 보면 QPushButton 에서 'clicked' 이벤트가 발생할 때 'app.quit' 메서드가 호출되도록 연결하는 코드를 명확히 이해할 수 있습니다. 다만 이러한 코딩 스타일은 app 이라는 변수가 여러 곳에서 사용될 수 있기 때문에 프로그램의 유지보수 관점에서 볼 때 좋은 코딩 스타일은 아닙니다.

2) QLabel

QLabel 위젯은 텍스트나 이미지를 출력하는 데 사용합니다. 그림 16.27 은 QLabel 과 QPushButton 위젯을 통해 생성한 윈도우입니다. 사용자가 'Click'이라는 버튼을 클릭하면 QLabel 위젯을 통해 '버튼이 클릭되었습니다.'라는 메시지를 출력합니다. 사용자가 'Clear' 버튼을 클릭하면 QLabel 위젯에 출력한 메시지가 지워집니다.

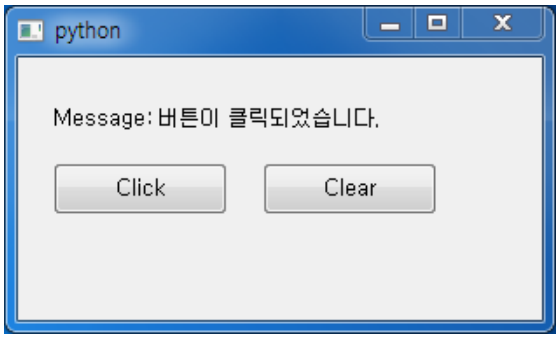


그림 16.27 QLabel 을 이용한 텍스트 출력

예제 16.6 QLabel 을 이용한 텍스트 출력(소스코드: book/ch16/08.py)

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        self.setGeometry(800, 400, 300, 150)

        textLabel = QLabel("Message: ", self)
        textLabel.move(20, 20)

        self.label = QLabel("", self)
        self.label.move(80, 20)
        self.label.resize(150, 30)

        btn1 = QPushButton("Click", self)
        btn1.move(20, 60)
        btn1.clicked.connect(self.btn1_clicked)

        btn2 = QPushButton("Clear", self)
        btn2.move(140, 60)
        btn2.clicked.connect(self.btn2_clicked)

    def btn1_clicked(self):
        self.label.setText("버튼이 클릭되었습니다.")

    def btn2_clicked(self):
        self.label.clear()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

예제 16.6 은 그림 16.27 프로그램에 대한 코드입니다. 이벤트 루프를 생성하고 MyWindow 클래스를 정의한 부분은 이전 예제와 같습니다. 다만 setupUI 메서드에서 위젯을 생성하고 시그널과 슬롯을 연결하는 부분과 시그널 처리를 위한 슬롯 구현이 수정됐습니다.

그림 16.27 을 참조하면 화면에 텍스트를 출력하는 **QLabel 객체가 두 개** 사용됐습니다. **윈도우에서 항상 'Message'라는 텍스트를 출력하는 객체는 textLabel 이라는 변수로 바인딩**합니다. 이와 달리 **버튼을 클릭할 때 '버튼이 클릭되었습니다.'라는 메시지를 출력하는 QLabel 객체는 self.label 이라는 변수로 바인딩**하고 있습니다.

어떤 객체에는 self 를 붙이고 어떤 객체에는 self 를 붙이지 않고 바인딩합니다. 저자가 십여 년 전에 프로그래밍을 처음 공부할 때는 이 부분이 매우 이해하기 어려웠습니다. 생성한 위젯을 클래스 내의 다른 메서드(setupUI 가 아닌)에서 참조할 때는 변수 이름에 self 를 붙여야 하고 그렇지 않은 경우에는 self 를 붙이지 않는다고 생각하면 쉽습니다.

예제 16.6 에서 self.label 은 setupUI 메서드뿐 아니라 btn1_clicked 와 btn2_clicked 라는 메서드에서도 사용됩니다. 따라서 self 를 붙여서 변수 이름을 생성했습니다. 이와 달리 textLabel 은 setupUI 메서드에서만 사용하고 더는 참조되지 않기 때문에 self 를 붙이지 않아도 되는 것입니다.

QLabel 클래스의 인스턴스를 생성한 후에는 move 메서드를 통해 해당 객체가 출력되는 위치를 조정할 수 있고, resize 메서드 호출을 통해 크기를 조정할 수 있습니다.

그림 16.27 에서 윈도우 하단에는 두 개의 버튼이 존재합니다. 'Click'이라고 표시된 버튼을 클릭하면 QLabel 객체에 '버튼이 클릭되었습니다.'라는 텍스트가 출력됩니다. 'Clear'라고 표시된 버튼을 클릭하면 QLabel 객체에 출력된 텍스트가 지워집니다. 버튼 클릭이라는 이벤트를 적절히 처리해주기 위해 각 버튼 위젯의 'clicked'라는 시그널을 self.btn_clicked, self.btn2_clicked 라는 슬롯과 각각 연결했습니다.

btn1 객체에서 'clicked' 시그널이 발생하는 경우 btn1_clicked 라는 슬롯(메서드)이 호출됩니다. 따라서 해당 메서드 내에서 QLabel 객체에 텍스트를 출력하면 됩니다. QLabel 객체를 바인딩하고 있는 self.label 변수로 setText 메서드를 호출해서 텍스트를 QLabel 객체에 출력하면 됩니다.

```
def btn1_clicked(self):
    self.label.setText("버튼이 클릭되었습니다.")
```

btn2 객체에서 'clicked'라는 시그널이 발생하면 QLabel 객체에 출력된 문자열을 지웁니다. 이 경우 역시 btn1 객체와 마찬가지로 self.label 변수를 통해 QLabel 객체에 접근한 후 clear 메서드를 호출해서 문자열을 지울 수 있습니다.

```
def btn2_clicked(self):
    self.label.clear()
```

3) QLineEdit 와 QStatusBar

QLineEdit 는 한 줄의 텍스트를 입력할 수 있는 위젯으로, 사용자로부터 간단한 텍스트를 입력받을 때 사용합니다. 그림 16.28 은 QLineEdit 를 이용해 사용자로부터 주식 종목 코드를 입력받는 프로그램입니다.

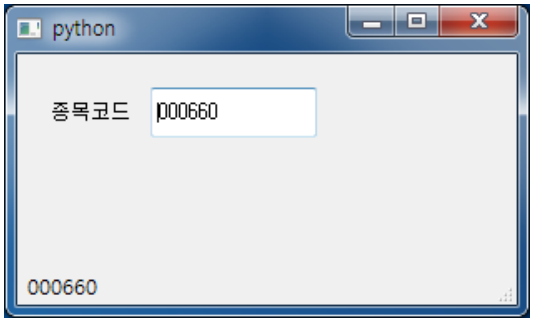


그림 16.28 QLineEdit 사용 예

예제 16.7 QLineEdit 사용 예(소스코드: book/ch16/09.py)

예제 16.7 은 그림 16.28 에 나온 프로그램의 소스코드입니다.

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        self.setGeometry(800, 400, 300, 150)

        # Label
        label = QLabel("종목코드", self)
        label.move(20, 20)

        # LineEdit
        self.lineEdit = QLineEdit("", self)
        self.lineEdit.move(80, 20)
        self.lineEdit.textChanged.connect(self.lineEditChanged)

        # StatusBar
        self.statusBar = QStatusBar(self)
        self.setStatusBar(self.statusBar)

    def lineEditChanged(self):
        self.statusBar.showMessage(self.lineEdit.text())

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

여섯개의 숫자로 구성된종목코드는 QLineEdit 를 이용해 사용자로부터 입력받을 수 있습니다. 이때 중요한 점은 QLineEdit 객체의 시그널 중 적당한 시그널을 사용해 사용자 이벤트를 처리해야 한다는 점입니다.

표 16.1 은 QLineEdit 위젯에서 자주 사용되는 시그널을 나타냅니다. textChanged 는 사용자의 입력으로 텍스트가 변경되면 발생합니다. returnPressed 는 사용자가 텍스트를 입력한 후 QLineEdit 객체에 서 엔터키를 눌렀을때 발생합니다.

표 16.1 자주 사용되는 QLineEdit 시그널

시그널	시그널 발생 시점
textChanged( )	QLineEdit 객체에서 텍스트가 변경될 때 발생
returnPressed( )	QLineEdit 객체를 통해 사용자가 엔터키를 눌렀을 때

예제 16.7 에서는 사용자가 종목 코드를 입력하는 순간에 시그널을 발생시키기 위해 textChanged 를 사용했습니다. 다음과 같이 textChanged 라는 시그널과 self.lineEditChanged 라는 슬롯을 연결하면 사용자가 종목 코드를 입력할 때 self.lineEditChanged 라는 메서드가 계속해서 호출됩니다.

```
self.lineEdit = QLineEdit("", self)
self.lineEdit.move(80, 20)
self.lineEdit.textChanged.connect(self.lineEditChanged)
```

사용자가 입력한 종목 코드를 다시 사용자에게 보여주기 위한 용도로 QStatusBar 위젯을 생성했습니다.

```
# StatusBar
self.statusBar = QStatusBar(self)
self.setStatusBar(self.statusBar)
```

lineEditChanged 메서드에서는 self.statusBar 라는 변수로 showMessage 메서드를 호출함으로써 상태 표시줄에 텍스트를 출력할 수 있습니다. 이때 출력될 문자열은 사용자가 QLineEdit 객체에 입력한 텍스트입니다. QLineEdit 객체에 입력된 문자열을 구하기 위해 self.lineEdit 변수를 통해 QLineEdit 객체에 접근한 후 text 메서드를 통해 사용자가 입력한 문자열을 얻어옵니다.

```
def lineEditChanged(self):
    self.statusBar.showMessage(self.lineEdit.text())
```

4) QRadioButton 과 QGroupBox

이번 절에서는 QRadioButton 과 QGroupBox 위젯에 대해 알아보도록 하겠습니다. QRadioButton 은 사용자로부터 여러 가지 옵션 중 하나를 입력받을 때 주로 사용하며, QGroupBox 는 제목이 있는 네모 박스 형태의 경계선을 만드는 데 사용합니다.

그림 16.29 에서 일봉, 주봉, 월봉이 QRadioButton 위젯 객체에 해당하고 '시간 단위'로 표시된 네모 박스가 QGroupBox 위젯 객체입니다. 보통 그림 16.29 와 같이 QRadioButton 과 QGroupBox 위젯을 동시에 사용하는 경우가 많습니다. 그림 16.29 의 프로그램은 사용자가 QRadioButton 위젯 중 하나를 선택할 때 QStatusBar 객체에 텍스트를 출력합니다.

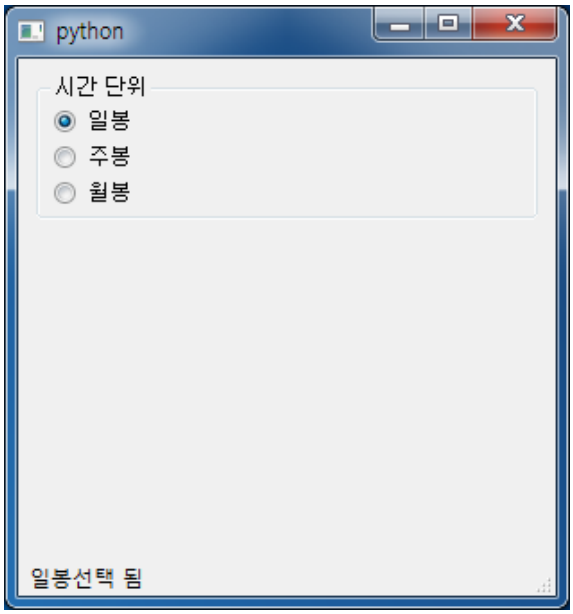


그림 16.29 QRadioButton 과 QGroupBox 사용 예

예제 16.8 QRadioButton, QGroupBox 사용 예(소스 코드: book/ch16/10.py)

예제 16.8 은 그림 16.29 에 나온 프로그램의 전체 코드입니다.

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        self.setGeometry(800, 200, 300, 300)

        groupBox = QGroupBox("시간 단위", self)
        groupBox.move(10, 10)
        groupBox.resize(280, 80)

        self.radio1 = QRadioButton("일봉", self)
        self.radio1.move(20, 20)
        self.radio1.setChecked(True)
        self.radio1.clicked.connect(self.radioButtonClicked)

        self.radio2 = QRadioButton("주봉", self)
        self.radio2.move(20, 40)
        self.radio2.clicked.connect(self.radioButtonClicked)
```

```
        self.radio3 = QRadioButton("월봉", self)
        self.radio3.move(20, 60)
        self.radio3.clicked.connect(self.radioButtonClicked)

        self.statusBar = QStatusBar(self)
        self.setStatusBar(self.statusBar)

    def radioButtonClicked(self):
        msg = ""
        if self.radio1.isChecked():
            msg = "일봉"
        elif self.radio2.isChecked():
            msg = "주봉"
        else:
            msg = "월봉"
        self.statusBar.showMessage(msg + "선택 됨")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

그림 16.29 의 UI 를 구성하는 코드부터 살펴보겠습니다. 먼저 QGroupBox 위젯을 이용해 제목이 '시간 단위'인 네모 박스를 만듭니다. 이때 출력 위치를 설정하기 위해 move 메서드를 호출했고 크기를 조절하기 위해 resize 메서드를 호출했습니다. 참고로 resize 메서드의 첫 번째 인자는 위젯의 너비(width)이고 두 번째 인자는 높이(height)입니다.

```
groupBox = QGroupBox("시간 단위", self)
groupBox.move(10, 10)
groupBox.resize(280, 80)
```

라디오 버튼도 다른 위젯과 비슷하게 QRadioButton 클래스의 생성자를 호출함으로써 객체를 생성할 수 있습니다. 첫 번째 인자로 출력될 문자열을 전달하고, 두 번째 인자로 위젯이 출력될 부모 위젯(parent widget)을 전달합니다.

라디오 버튼 역시 move 메서드를 통해 위젯의 출력 위치를 설정할 수 있습니다. setChecked 메서드는 라디오 버튼의 초기 상태를 설정할 수 있습니다. True 로 설정하는 경우 해당 라디오 버튼은 선택된 상태로 출력됩니다.

```
self.radio1 = QRadioButton("일봉", self)
self.radio1.move(20, 20)
self.radio1.setChecked(True)
```

라디오 버튼도 'clicked'라는 시그널을 가집니다. 따라서 라디오 버튼이 클릭됐을 때 호출되는 슬롯을 설정할 수 있습니다.

```
self.radio1.clicked.connect(self.radioButtonClicked)
```

그림 16.29 에는 총 3 개의 라디오 버튼이 있는데, 세 개의 라디오 버튼 객체는 모두 self.radioButtonClicked 라는 동일한 슬롯을 사용하고 있습니다. 따라서 사용자가 세 개의 라디오 버튼 중 하나를 클릭하면 self.radioButtonClicked 라는 슬롯이 호출됩니다.

self.radioButtonClicked 슬롯에는 사용자가 '일봉', '주봉', '월봉'에 해당하는 라디오 버튼을 클릭하면 선택된 시간 단위가 QStatusBar 객체에 출력되도록 구현합니다. QStatusBar 에 텍스트를 출력하는 데는 showMessage 메서드를 사용했습니다.

세 개의 라디오 버튼 모두가 self.radioButtonClicked 라는 슬롯과 연결돼 있으므로 슬롯이 호출될 때 각 라디오 버튼 객체에 대해 isChecked 메서드를 호출해 어떤 라디오 버튼이 선택됐는지 확인해야합니다.

예를 들어, 사용자가 '일봉'으로 표시된 라디오 버튼을 선택하는 경우 해당 라디오 버튼 객체를 바인딩하고 있는 self.radio1 의 isChecked 메서드 호출의 결과가 True 를 리턴합니다. 따라서 다음과 같은 if 문을 사용해 어떤 라디오 버튼을 선택했는지 확인할 수 있습니다.

```
def radioButtonClicked(self):
    msg = ""
    if self.radio1.isChecked():
        msg = "일봉"
    elif self.radio2.isChecked():
        msg = "주봉"
    else:
        msg = "월봉"
    self.statusBar.showMessage(msg + "선택 됨")
```

5) QCheckBox

이번 절에서는 QRadioButton 과 비슷하게 사용자로부터 옵션을 선택받을 때 사용하는 QCheckBox 위젯을 알아보겠습니다. QCheckBox 는 여러 옵션 중 하나만 선택할 수 있었던 QRadioButton 과 달리 여러 옵션을 동시에 선택할 수도 있습니다.

그림 16.30 은 QCheckBox 를 사용해 이동평균선에 대한 옵션을 선택하는 프로그램입니다. 사용자는 '5 일 이동평균선', '20 일 이동평균선', '60 일 이동평균선'이라는 옵션에 대해 동시에 선택할 수 있습니다. QCheckBox 위젯과 QRadioButton 위젯 중 어떤 위젯을 사용할지는 프로그래머가 상황에 맞춰 선택하면 됩니다.

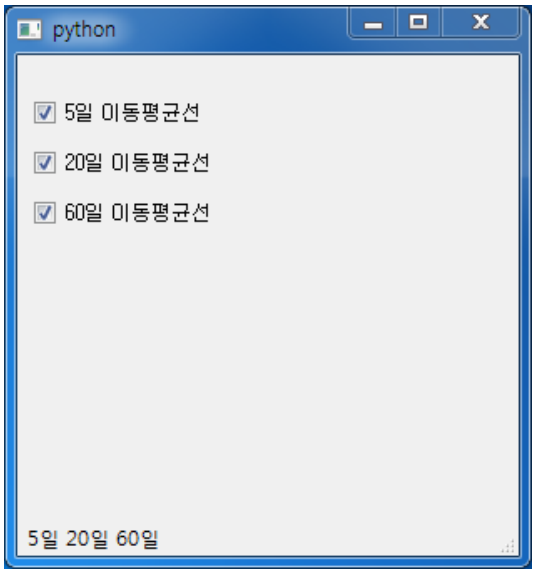


그림 16.30 QCheckBox

예제 16.9 QCheckBox 사용 예(소스코드: book/ch16/11.py)

예제 16.9 는 그림 16.30 에 나온 프로그램의 전체 코드입니다.

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        self.setGeometry(800, 200, 300, 300)

        self.checkBox1 = QCheckBox("5 일 이동평균선", self)
        self.checkBox1.move(10, 20)
        self.checkBox1.resize(150, 30)
        self.checkBox1.stateChanged.connect(self.checkBoxState)

        self.checkBox2 = QCheckBox("20 일 이동평균선", self)
        self.checkBox2.move(10, 50)
        self.checkBox2.resize(150, 30)
        self.checkBox2.stateChanged.connect(self.checkBoxState)

        self.checkBox3 = QCheckBox("60 일 이동평균선", self)
        self.checkBox3.move(10, 80)
        self.checkBox3.resize(150, 30)
        self.checkBox3.stateChanged.connect(self.checkBoxState)
```

```
        self.statusBar = QStatusBar(self)
        self.setStatusBar(self.statusBar)

    def checkBoxState(self):
        msg = ""
        if self.checkBox1.isChecked() == True:
            msg += "5 일 "
        if self.checkBox2.isChecked() == True:
            msg += "20 일 "
        if self.checkBox3.isChecked() == True:
            msg += "60 일 "
        self.statusBar.showMessage(msg)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

CheckBox 를 사용하려면 QCheckBox 클래스의 인스턴스를 생성합니다. 인스턴스를 생성할 때는 QCheckBox 에 표시될 텍스트와 QCheckBox 가 출력될 부모 위젯을 인자로 전달합니다. QCheckBox 는 QMainWindow 객체 안에 출력되기 때문에 부모 위젯으로 self 를 사용합니다.

QCheckBox 클래스는 QRadioButton 클래스와 마찬가지로 move 와 resize 라는 메서드를 가집니다. move 메서드를 통해 QCheckBox 위젯이 출력될 위치를 결정할 수 있고, resize 를 통해 크기를 지정할 수 있습니다.

```
self.checkBox1 = QCheckBox("5 일 이동평균선", self)
self.checkBox1.move(10, 20)
self.checkBox1.resize(150, 30)
```

'5 일 이동평균선'이라는 텍스트를 출력하는 QCheckBox 위젯을 self.checkBox1 이라는 인스턴스 변수가 바인딩하고 있습니다. 같은 방식으로 '20 일 이동평균선'과 '60 일 이동평균선'이라는 텍스트를 가진 CheckBox 를 생성합니다.

```
self.checkBox2 = QCheckBox("20 일 이동평균선", self)
self.checkBox2.move(10, 50)
self.checkBox2.resize(150, 30)

self.checkBox3 = QCheckBox("60 일 이동평균선", self)
self.checkBox3.move(10, 80)
self.checkBox3.resize(150, 30)
```

QCheckBox 가 선택될 때 어떤 QCheckBox 가 선택되었는가에 대한 정보를 출력할 용도로 QStatusBar 위젯도 생성합니다.

```
self.statusBar = QStatusBar(self)
self.setStatusBar(self.statusBar)
```

화면에 출력될 위젯을 모두 생성했으니 이번에는 각 위젯에서 발생하는 이벤트를 처리하기 위해 시그널과 슬롯을 연결하는 작업을 해보겠습니다.

앞서 설명한 것처럼 PyQt 의 위젯은 각 위젯의 특성에 맞는 시그널을 갖고 있습니다. QCheckBox 는 CheckBox 가 선택되지 않은 상태에서 선택되거나 반대로 선택된 상태에서 선택 해제가 된 경우에 stateChanged 라는 시그널을

발생시킵니다. 따라서 사용자가 QCheckBox 위젯의 상태를 변경할 때 이를 처리하는 함수인 슬롯을 stateChanged 시그널과 연결하면 됩니다.

```
self.checkBox1.stateChanged.connect(self.checkBoxState)
self.checkBox2.stateChanged.connect(self.checkBoxState)
self.checkBox3.stateChanged.connect(self.checkBoxState)
```

그림 16.30 에 나온 3 개의 QCheckBox 객체 중 하나라도 상태가 변경되면 checkBoxState 메서드가 호출됩니다. 따라서 checkBoxState 메서드가 호출될 때 어떤 QCheckBox 의 상태가 변경되어 이벤트가 발생했는지 알아내야 합니다.

QCheckBox 는 QRadioButton 과 마찬가지로 isChecked 메서드를 통해 QCheckBox 위젯 객체가 현재 선택된 상태인지 아닌지 확인할 수 있습니다. isChecked 메서드를 이용해 self.checkBox1, self.checkBox2, self.checkBox3 이 바인딩하고 있는 QCheckBox 객체 중 어떤 QCheckBox 가 선택됐는지 확인합니다. QCheckBox 위젯은 동시에 여러 개가 선택될 수도 있으므로 if, elif, else 같은 형태가 아니라 각 위젯에 대해 if 문으로 상태를 비교해야 합니다.

```
def checkBoxState(self):
    msg = ""
    if self.checkBox1.isChecked() == True:
        msg += "5 일 "
    if self.checkBox2.isChecked() == True:
        msg += "20 일 "
    if self.checkBox3.isChecked() == True:
        msg += "60 일 "
    self.statusBar.showMessage(msg)
```

6) QSpinBox

이번 절에서는 사용자로부터 정숫값을 입력받을 때 사용하는 QSpinBox 위젯에 대해 알아보겠습니다. QSpinBox 라는 위젯 이름에서 'spin'이라는 영어 단어는 '회전', '돌다.' 등의 의미를 가집니다.

그림 16.31 은 QSpinBox 를 사용한 프로그램입니다. QSpinBox 위젯을 보면 값을 증가시키는 데 사용하는 화살표, 값을 감소시킬 때 사용하는 화살표, 값이 출력되는 부분으로 구성됨을 알 수 있습니다. QSpinBox 에는 값을 증가/감소시키기 위한 화살표가 존재하기 때문에 사용자가 값을 입력할 때보다 편리하게 입력할 수 있습니다.

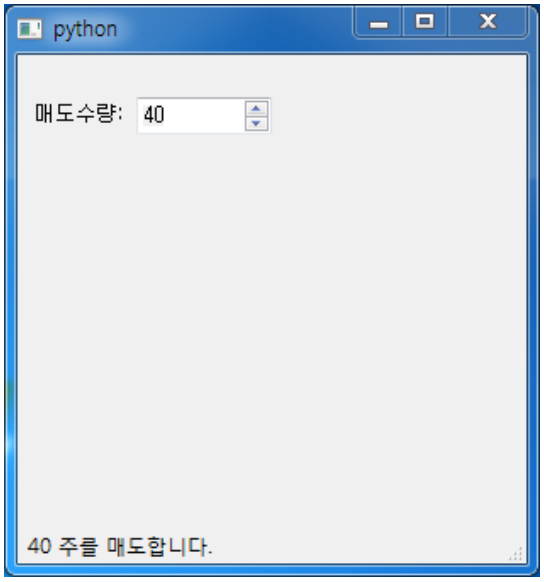


그림 16.31 QSpinBox

설명만으로는 위젯의 특성을 제대로 파악하기 어려우므로 예제 16.10 을 먼저 실행해 보기 바랍니다. QSpinBox 위젯을 이용하여 정숫값을 입력할 때는 QSpinBox 위젯에 직접 값을 입력하거나 초깃값으로부터 시작해 화살표를 통해서 값을 증가시키거나 감소시켜서 입력해도 됩니다.

예제 16.10 QSpinBox 사용 예(소스코드: book/ch16/12.py)

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        self.setGeometry(800, 200, 300, 300)

        label = QLabel("매도수량: ", self)
        label.move(10, 20)

        self.spinBox = QSpinBox(self)
        self.spinBox.move(70, 25)
        self.spinBox.resize(80, 22)
        self.spinBox.valueChanged.connect(self.spinBoxChanged)

        self.statusBar = QStatusBar(self)
        self.setStatusBar(self.statusBar)

    def spinBoxChanged(self):
        val = self.spinBox.value()
        msg = '%d 주를 매도합니다.' % (val)
        self.statusBar.showMessage(msg)
```

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

QSpinBox 위젯 동작에 대해서 어느 정도 이해되면 지금부터는 위젯을 구성하는 부분과 위젯이 제공하는 시그널에 대해서 알아보겠습니다.

QSpinBox 위젯 역시 다른 위젯과 마찬가지로 생성자를 호출해 QSpinBox 클래스에 대한 인스턴스를 생성함으로써 화면에 출력할 수 있습니다. QSpinBox 는 QCheckBox 와 달리 텍스트를 출력하지는 않기 때문에 생성자의 인자로 부모 위젯만 전달합니다. 현재 QSpinBox 는 QMainWindow 라는 위젯 위에 출력돼야 하므로 self 라는 값을 생성자의 인자로 전달하면 됩니다.

QSpinBox 위젯의 출력 위치와 크기 조절에는 다른 위젯과 마찬가지로 move 와 resize 라는 메서드를 사용합니다.

```
self.spinBox = QSpinBox(self)
self.spinBox.move(70, 25)
self.spinBox.resize(80, 22)
```

QSpinBox 위젯을 생성했다면 QSpinBox 가 제공하는 시그널에 대해 알아봅시다. QSpinBox 는 사용자로부터 값을 입력받기 때문에 값이 변할 때마다 값이 변경됐다는 사실을 프로그램에 알려줘야 합니다. QSpinBox 는 valueChanged 라는 시그널을 가지는데, 해당 시그널이 바로 QSpinBox 의 값이 변경될 때 자동으로 발생하는 시그널입니다.

값이 변경될 때 valueChanged 라는 시그널이 발생하므로 해당 시그널과 이를 처리할 슬롯을 연결하면 사용자 이벤트를 처리할 수 있습니다.

```
self.spinBox.valueChanged.connect(self.spinBoxChanged)
```

QSpinBox 위젯 값이 변경될 때 호출되는 메서드인 spinBoxChanged 에서는 현재 QSpinBox 위젯 값을 얻어와야 합니다. value 메서드를 호출해서 얻어온 값은 정수 타입이기 때문에 먼저 문자열로 변경한 후 QStatusBar 객체에 출력합니다.

```
def spinBoxChanged(self):
    val = self.spinBox.value()
    msg = '%d 주를 매도합니다.' % (val)
    self.statusBar.showMessage(msg)
```

QSpinBox 클래스는 value 메서드 외에도 몇 가지 유용한 메서드를 제공합니다. 대표적으로 많이 사용되는 메서드에 대해서만 간략히 살펴보겠습니다.

첫 번째는 QSpinBox 의 초깃값을 설정하는 것입니다. 예를 들어 매수 수량의 초깃값을 0 으로 설정하는 것보다 10 으로 설정해 주는 편이 사용자에게 더 편리할 수 있습니다.

두 번째는 QSpinBox 의 화살표를 눌렀을 때 증가 또는 감소하는 값을 설정하는 것입니다. 유가증권시장이나 코스닥시장에서 소형주는 1 주 단위로도 거래할 수 있지만 대형주는 10 주 단위로 거래할 수도 있습니다. 따라서 QSpinBox 의 화살표를 눌렀을 때 증가 또는 감소하는 값을 지정해두면 사용자가 더욱 쉽게 값을 입력할 수 있습니다.

마지막은 QSpinBox 를 통해 입력할 수 있는 값의 범위를 지정해주는 것입니다. 매도할 때 0 주를 매도하는 것은 불가능하므로 매도 수량의 최솟값은 0 보다 클 것입니다. 사용하는 시나리오에 따라서 최솟값과 최댓값을 지정해주면 사용자가 보다 편리하게 값을 입력할 수 있습니다.

QSpinBox 클래스에서 초깃값의 설정은 setValue 메서드를 사용합니다. QSpinBox 에서 화살표를 눌렀을 때 증가 또는 감소하는 값은 setSingleStep 메서드를 사용합니다. 마지막으로 최솟값과 최댓값을 설정할 때는 setMinimum 과 setMaximum 메서드를 사용합니다.

예제 16.10 에서 QSpinBox 와 관련된 코드를 다음과 같이 수정해주면 QSpinBox 가 10 부터 시작해서 10 단위로 증가 또는 감소하는 것을 확인할 수 있습니다. 지금 설명한 부분은 어쩌면 프로그래머에게는 지루한 작업일 수 있습니다. 하지만 이러한 사소한 배려가 프로그램을 사용하는 사용자가 느끼는 UI/UX 에 큰 차이를 줍니다.

```
self.spinBox = QSpinBox(self)
self.spinBox.move(70, 25)
self.spinBox.resize(80, 22)

self.spinBox.setValue(10)
self.spinBox.setSingleStep(10)
self.spinBox.setMinimum(1)
self.spinBox.setMaximum(10000)
self.spinBox.valueChanged.connect(self.spinBoxChanged)
```

7) QTableWidget

이번 절에서는 그림 16.32 에서 사용한 QTableWidget 에 대해 알아보겠습니다. 일상생활에서 그림 16.32 와 같이 행과 열로 구성된 2 차원 포맷 형태의 데이터를 많이 사용합니다. 여러분이 자주 사용하는 마이크로소프트의 엑셀도 데이터를 2 차원 포맷 형태로 표현하는 대표적인 프로그램입니다. 이처럼 데이터를 행과 열로 표시하면 데이터를 쉽게 파악하는 데 도움이 됩니다.

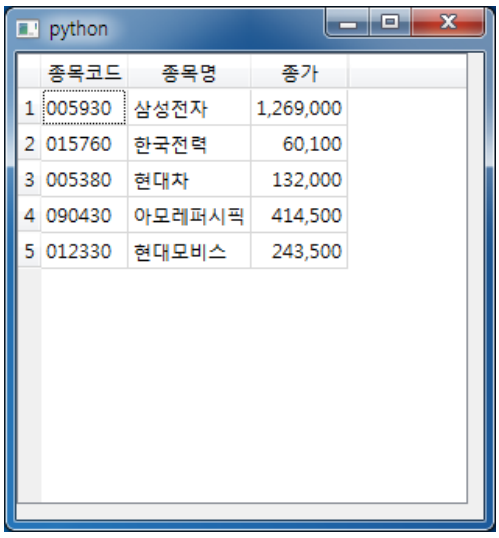


그림 16.32 QTableWidget

먼저 그림 16.33 과 같이 간단한 형태의 프로그램을 작성하면서 QTableWidget 클래스의 주요 기능을 파악해 보겠습니다.

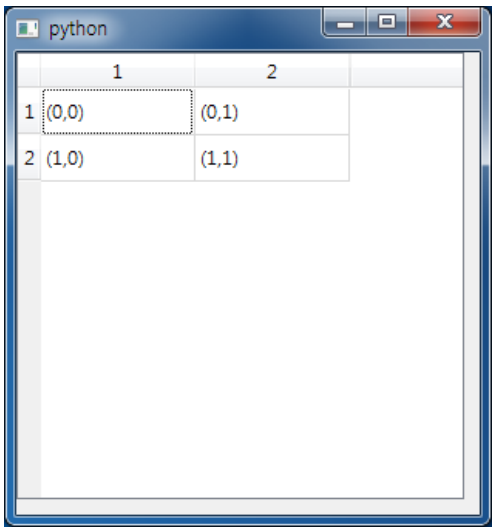


그림 16.33 QTableWidget – 2

예제 16.11 QTableWidget 사용 예(소스코드: book/ch16/14.py)

예제 16.11 은 그림 16.33 에 나온 프로그램의 전체 코드입니다.

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setupUI()
```

```
    def setupUI(self):
        self.setGeometry(800, 200, 300, 300)

        self.tableWidget = QTableWidget(self)
        self.tableWidget.resize(290, 290)
        self.tableWidget.setRowCount(2)
        self.tableWidget.setColumnCount(2)
        self.setTableWidgetData()

    def setTableWidgetData(self):
        self.tableWidget.setItem(0, 0, QTableWidgetItem("(0,0)"))
        self.tableWidget.setItem(0, 1, QTableWidgetItem("(0,1)"))
        self.tableWidget.setItem(1, 0, QTableWidgetItem("(1,0)"))
        self.tableWidget.setItem(1, 1, QTableWidgetItem("(1,1)"))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

먼저 setupUI 메서드에서 QTableWidget 클래스의 인스턴스를 생성하는 코드를 살펴보겠습니다. QTableWidget 클래스의 인스턴스를 생성하기 위해 생성자를 호출할 때 QTableWidget 클래스가 표시될 부모 클래스를 지정합니다.

QTableWidget 클래스의 크기는 resize 메서드를 이용해 조절할 수 있습니다. QTableWidget 클래스는 데이터를 출력하기에 앞서 행과 열의 개수를 미리 지정해야 합니다. 행의 개수를 지정할 때는 setRowCount 메서드를 사용하고, 열의 개수를 지정할 때는 setColumnCount 메서드를 사용합니다.

```
self.tableWidget = QTableWidget(self)
self.tableWidget.resize(290, 290)
self.tableWidget.setRowCount(2)
self.tableWidget.setColumnCount(2)
```

2x2 (행 x 열) 크기의 QTableWidget 객체가 생성되면 (0, 0), (0, 1), (1, 0), (1, 1) 위치에 아이템을 넣을 수 있습니다. 각 위치에는 setItem 메서드를 사용하여 QTableWidgetItem 객체를 삽입하면 됩니다.

setItem 메서드는 3 개의 인자를 받는데 첫 번째 인자는 행에 대한 인덱스이고, 두 번째 인자는 열에 대한 인덱스입니다. 마지막으로 세 번째 인자는 QTableWidgetItem 객체를 전달합니다.

```
def setTableWidgetData(self):
    self.tableWidget.setItem(0, 0, QTableWidgetItem("(0,0)"))
    self.tableWidget.setItem(0, 1, QTableWidgetItem("(0,1)"))
    self.tableWidget.setItem(1, 0, QTableWidgetItem("(1,0)"))
    self.tableWidget.setItem(1, 1, QTableWidgetItem("(1,1)"))
```

예제 16.12 QTableWidget 사용 예 – 2(소스코드: book/ch16/15.py)

QTableWidget 클래스에 대해 간단히 알아봤으니 이번에는 그림 16.32 에 나오느 프로그램의 코드를 살펴보겠습니다.

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

kospi_top5 = {
    'code': ['005930', '015760', '005380', '090430', '012330'],
```

```
    'name': ['삼성전자', '한국전력', '현대차', '아모레퍼시픽', '현대모비스'],
    'cprice': ['1,269,000', '60,100', '132,000', '414,500', '243,500']
}
column_idx_lookup = {'code': 0, 'name': 1, 'cprice': 2}

class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        self.setGeometry(800, 200, 300, 300)

        self.tableWidget = QTableWidget(self)
        self.tableWidget.resize(290, 290)
        self.tableWidget.setRowCount(5)
        self.tableWidget.setColumnCount(3)
        self.tableWidget.setEditTriggers(QAbstractItemView.NoEditTriggers)

        self.setTableWidgetData()

    def setTableWidgetData(self):
        column_headers = ['종목코드', '종목명', '종가']
        self.tableWidget.setHorizontalHeaderLabels(column_headers)

        for k, v in kospi_top5.items():
            col = column_idx_lookup[k]
            for row, val in enumerate(v):
                item = QTableWidgetItem(val)
                if col == 2:
                    item.setTextAlignment(Qt.AlignVCenter | Qt.AlignRight)

                self.tableWidget.setItem(row, col, item)

        self.tableWidget.resizeColumnsToContents()
        self.tableWidget.resizeRowsToContents()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

먼저 UI 를 구성하는 코드를 살펴보면 예제 16.11 과 크게 다르지는 않습니다. 다만 QTableWidget 의 행과 열의 개수가 다르고 setEditTriggers 메서드를 사용해 사용자가 QTableWidget 의 아이템 항목을 수정할 수 없도록 설정했습니다.

```
self.tableWidget = QTableWidget(self)
self.tableWidget.resize(290, 290)
self.tableWidget.setRowCount(5)
self.tableWidget.setColumnCount(3)
self.tableWidget.setEditTriggers(QAbstractItemView.NoEditTriggers)
```

QTableWidget 에 아이템을 채우는 코드를 살펴보겠습니다. 먼저 column 에 대한 라벨을 설정했습니다. 참고로 row 방향에 대한 라벨을 설정할 때는 setVerticalHeaderLabels 메서드를 사용합니다.

```
column_headers = ['종목코드', '종목명', '종가']
self.tableWidget.setHorizontalHeaderLabels(column_headers)
```

5x3 크기의 QTableWidget 에 아이템을 채우는 코드는 (행, 열) 위치에 대한 인덱스를 구하고 해당 위치에 문자열을 넣는 방식으로 구현했습니다. 이때 열(column)에 대한 인덱스는 미리 정의된 column_idx_lookup 이라는 딕셔너리를 사용해 계산합니다.

QTableWidget 에 아이템으로 삽입하려면 먼저 데이터를 QTableWidgetItem 객체로 만들어야 합니다. 그런 다음 setItem 메서드를 이용해 아이템을 원하는 위치에 삽입하면 됩니다. 이때 '종가'에 해당하는 열 데이터는 setTextAlignment 메서드를 사용해 우측 정렬했습니다.

```
for k, v in kospi_top5.items():
    col = column_idx_lookup[k]
    for row, val in enumerate(v):
        item = QTableWidgetItem(val)
        if col == 2:
            item.setTextAlignment(Qt.AlignVCenter | Qt.AlignRight)
        self.tableWidget.setItem(row, col, item)
```

마지막으로 QTableWidget 의 행과 열 크기를 각 위치에 저장된 아이템 길이에 맞춰 조정합니다.

```
self.tableWidget.resizeColumnsToContents()
self.tableWidget.resizeRowsToContents()
```

4. Layout

GUI 프로그래밍에서 위젯을 배치하는 것을 레이아웃(layout)이라고 합니다. 앞서 여러분이 작성했던 코드에서는 resize 메서드를 사용하여 위젯의 크기를 변경했고, move 나 setGeometry 메서드를 이용해 위젯의 출력 위치를 설정했습니다.

그림 16.34 는 resize 와 move 메서드로 직접 위치와 크기를 설정해 레이아웃을 조정한 프로그램의 예입니다.

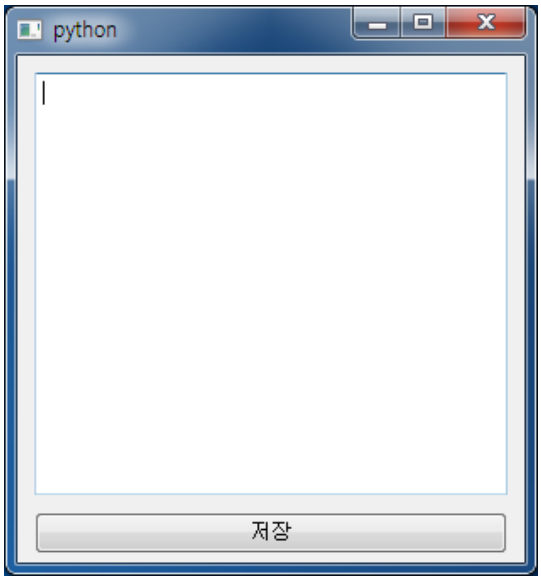


그림 16.34 위치 및 크기 지정을 통한 레이아웃

예제 16.13 위치 및 크기 설정을 통한 레이아웃(소스코드: book/ch16/16.py)

예제 16.13 은 그림 16.34 에 나오느 프로그램의 전체 코드입니다. QTextEdit 와 QPushButton 위젯의 위치와 크기를 직접 설정하는 것을 확인할 수 있습니다. 참고로 MyWindow 클래스를 정의할 때 QMainWindow 클래스 대신 **QWidget** 클래스를 상속했습니다.

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        self.setGeometry(800, 200, 300, 300)

        self.textEdit = QTextEdit(self)
        self.textEdit.resize(280, 250)
        self.textEdit.move(10, 10)

        self.pushButton= QPushButton('저장', self)
        self.pushButton.resize(280, 25)
        self.pushButton.move(10, 270)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
```

```
    app.exec_()
```

예제 16.13 과 같은 방식으로 레이아웃을 구성하면 사용자가 윈도우의 크기를 변경할 때 문제가 발생합니다. 예를 들어, 그림 16.35 는 사용자가 윈도우를 오른쪽 아래 방향으로 드래그해서 크기를 변경한 것입니다. 윈도우 크기는 변경됐지만 윈도우 내부에 존재하는 위젯의 출력 위치 및 크기는 변경되지 않기 때문에 전체적인 UI 구성이 어색해 보입니다.

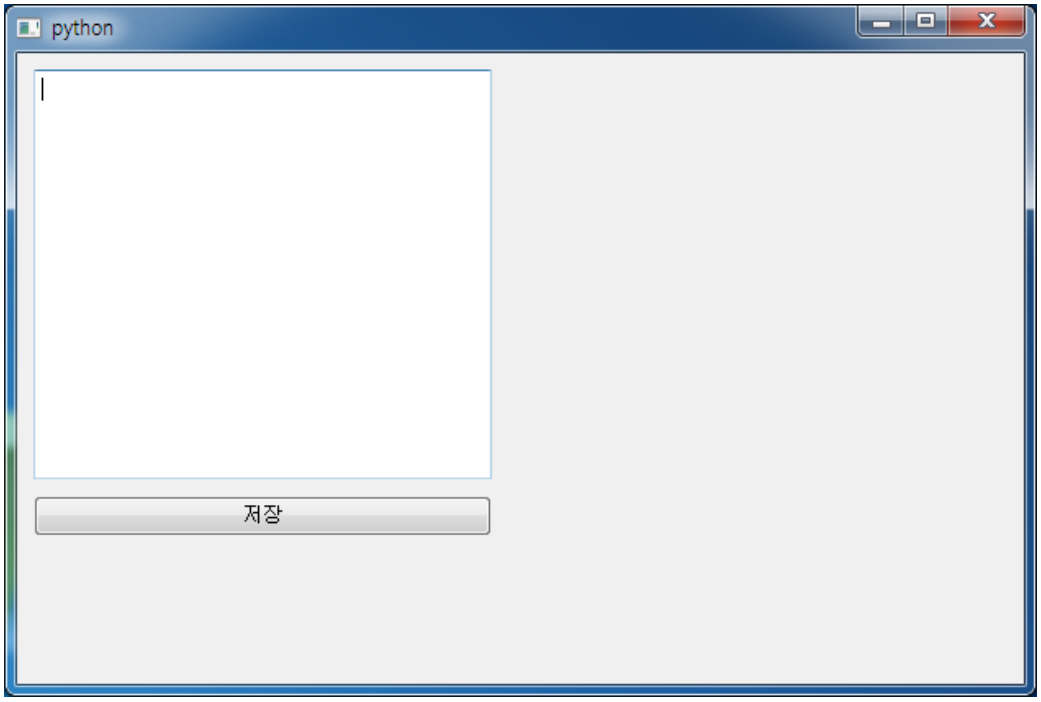


그림 16.35 잘못된 레이아웃의 예

이러한 문제는 PyQt 에서만 있는 것이 아니고 GUI 프로그래밍에서 공통으로 발생하는 문제입니다. 이러한 문제를 해결하는 고전적인 방법은 윈도우의 크기가 변경될 때 발생하는 시그널과 이를 처리하는 슬롯을 연결한 후, 해당 슬롯 내에서 윈도우 내부의 위젯의 크기와 위치를 다시 계산한 후 재설정하는 것입니다. 그러나 이러한 방식은 프로그래머에게 매우 귀찮은 작업입니다

1) QVBoxLayout

앞서 위젯의 크기와 출력 위치를 명시적으로 설정하는 방법은 윈도우 크기가 바뀔 때 문제가 있음을 확인했습니다. PyQt 에서는 이러한 문제를 쉽게 해결하고자 레이아웃 매니저를 제공합니다. PyQt 는 레이아웃 매니저로 QVBoxLayout, QHBoxLayout, QBoxLayout, QGridLayout, QLayout 이라는 다섯 가지 클래스를 제공합니다. 이번 절에서는 QVBoxLayout 클래스에 대해 먼저 알아보겠습니다.

QVBoxLayout 클래스는 이름을 통해 짐작할 수 있듯이 위젯을 수직 방향으로 나열합니다. 그림 16.34 를 참조하면 QTextEdit 위젯이 윈도우 상단에 위치하고 QPushButton 위젯이 QTextEdit 아래쪽에 위치합니다. 이러한 형태의 레이아웃일 때 QVBoxLayout 클래스를 사용합니다. 참고로 QVBoxLayout 을 사용하면 앞에서 살펴본 위젯의 크기 변경 문제가 자동으로 해결됩니다.

예제 16.14 QVBoxLayout 을 통한 레이아웃(소스코드: book/ch16/17.py)

예제 16.14 는 QVBoxLayout 클래스를 사용하여 그림 16.34 에 나온 프로그램의 레이아웃을 조정한 것입니다. 예제 16.13 과 달리 QTextEdit 와 QPushButton 위젯을 생성할 때 부모 위젯을 명시적으로 전달하지 않으며, resize 나 move 메서드도 호출하지 않는 것을 확인할 수 있습니다. 그 대신 QTextEdit 와 QPushButton 위젯을 QVBoxLayout 객체에 추가한 후 현재 윈도우의 레이아웃으로 QVBoxLayout 객체로 설정했습니다.

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        self.setGeometry(800, 200, 300, 300)

        self.textEdit = QTextEdit()
        self.pushButton= QPushButton('저장')

        layout = QVBoxLayout()
        layout.addWidget(self.textEdit)
        layout.addWidget(self.pushButton)

        self.setLayout(layout)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

예제 16.14 를 실행한 후 화면에 출력된 윈도우의 크기를 변경해 보기 바랍니다. 이번에는 그림 16.36 과 같이 윈도우의 크기를 자유롭게 변경해도 내부 위젯이 일정한 비율을 가지며 크기가 자동으로 바뀌는 것을 확인할 수 있습니다.

그림 16.36 QVBoxLayout 을 이용한 레이아웃 구성

2) QHBoxLayout

이번 절에서 다룰 QHBoxLayout 은 행(row) 방향으로 위젯을 배치할 때 사용하는 레이아웃 매니저입니다. QHBoxLayout 을 이용하면 그림 16.37 과 같은 형태의 UI 를 효과적으로 구성할 수 있습니다.

그림 16.37 QHBoxLayout 을 이용한 레이아웃 구성

예제 16.15 QHBoxLayout 을 이용한 레이아웃 구성(소스코드: book/ch16/18.py)

예제 16.15 는 그림 16.37 에 나온 프로그램의 전체 코드입니다.

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
```

```
        self.setGeometry(800, 200, 300, 100)

        self.pushButton1= QPushButton("Button1")
        self.pushButton2= QPushButton("Button2")
        self.pushButton3= QPushButton("Button3")

        layout = QHBoxLayout()
        layout.addWidget(self.pushButton1)
        layout.addWidget(self.pushButton2)
        layout.addWidget(self.pushButton3)

        self.setLayout(layout)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

예제 16.15 를 살펴보면 1) 위젯 만들기 2) 레이아웃 매니저 만들기 3) addWidget 메서드를 이용한 위젯 등록 4) 레이아웃 매니저 등록 단계로 구성돼 있습니다. 이는 앞에서 배운 QVBoxLayout 과 동일하며 앞으로 다룰 QGridLayout 도 같습니다.

레이아웃 매니저를 사용할 때 주의할 점은 레이아웃 매니저에 추가되는 위젯을 생성할 때는 부모 위젯을 지정할 필요가 없으며, 위젯의 크기나 위치를 명시적으로 설정하지 않아야 한다는 점입니다.

3) QGridLayout

QGridLayout 클래스는 격자 형태의 UI 를 구성하는 데 사용합니다. 예를 들어, 그림 16.38 에서는 2x3 (행 x 열) 크기의 격자 구조에 QLabel, QLineEdit, QPushButton 위젯을 배치했습니다. QGridLayout 은 QHBoxLayout 이나 QVBoxLayout 과 달리 격자에서 위젯을 배치할 좌표를 입력받습니다.

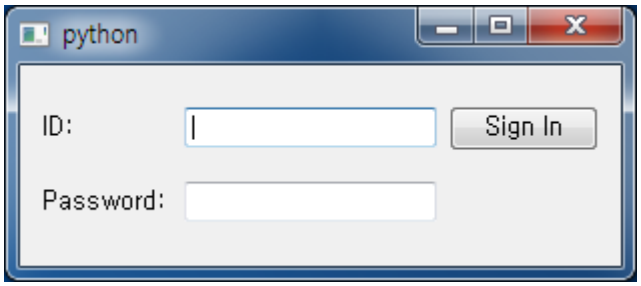


그림 16.38 QGridLayout 을 이용한 레이아웃 구성

예제 16.16 QGridLayout 을 이용한 레이아웃 구성(소스코드: book/ch16/19.py)

예제 16.16 은 그림 16.38 에 나온 프로그램의 전체 코드입니다. 앞에서 여러 번 설명한 것처럼 코드의 구성은 1) 위젯 만들기 2) 레이아웃 매니저 만들기 3) addWidget 메서드를 이용한 위젯 등록 4) 레이아웃 매니저 등록 단계로 구성됩니다. 다만 QGridLayout 에서는 addWidget 메서드에서 행과 열의 인덱스를 차례로 입력받습니다.

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QWidget):
    def __init__(self):
        super().__init__()
```

```
        self.setupUI()

    def setupUI(self):
        self.setGeometry(800, 200, 300, 100)

        self.label1 = QLabel("ID: ")
        self.label2 = QLabel("Password: ")
        self.lineEdit1 = QLineEdit()
        self.lineEdit2 = QLineEdit()
        self.pushButton1= QPushButton("Sign In")

        layout = QGridLayout()

        layout.addWidget(self.label1, 0, 0)
        layout.addWidget(self.lineEdit1, 0, 1)
        layout.addWidget(self.pushButton1, 0, 2)

        layout.addWidget(self.label2, 1, 0)
        layout.addWidget(self.lineEdit2, 1, 1)

        self.setLayout(layout)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

그림 16.39 는 그림 16.38 에서 사용한 2x3 (행 x 열) 격자를 그림으로 표현한 것입니다. 각 행과 열에 대해 0 부터 시작하는 정숫값을 통해 인덱싱하면 됩니다.

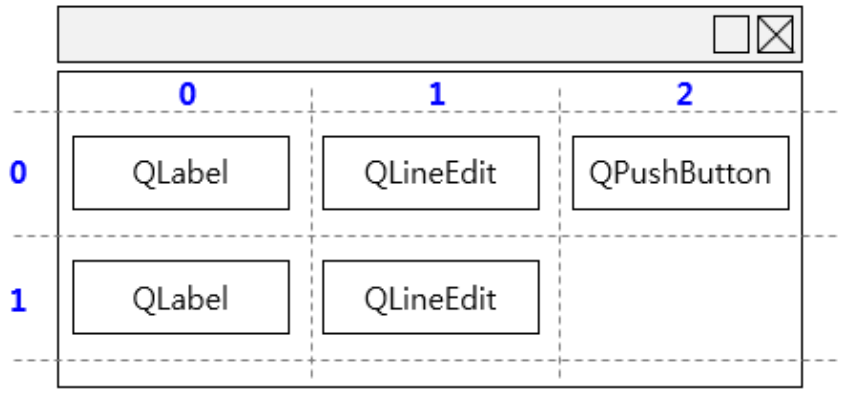


그림 16.39 QGridLayout 에서의 격자 셀

4) 레이아웃 중첩

이번 절에서는 QVBoxLayout 과 QHBoxLayout 을 중첩해서 그림 16.40 과 같이 조금 복잡한 형태의 레이아웃을 구성해 보겠습니다. 레이아웃 매니저 클래스에서 addWidget 메서드를 통해 레이아웃에 위젯을 추가할 수 있는 것처럼, **addLayout** 메서드를 호출하면 레이아웃에 다른 레이아웃 매니저를 추가할 수 있습니다.

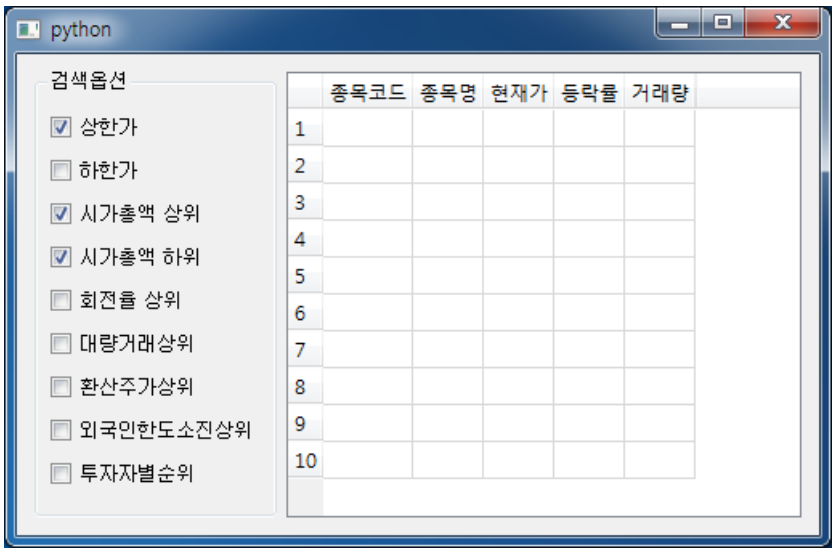


그림 16.40 레이아웃 중첩

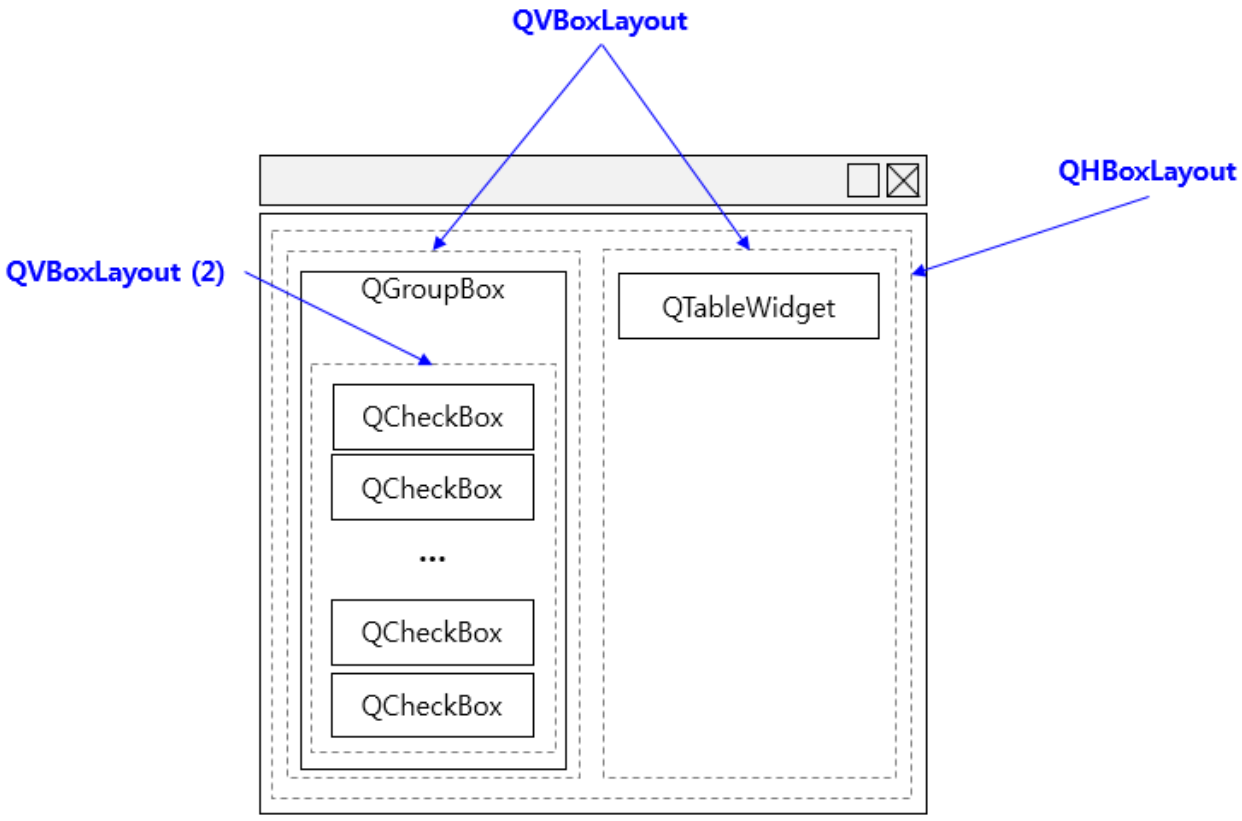


그림 16.41 레이아웃 중첩을 통한 UI 구성

그림 16.41 은 그림 16.40 의 프로그램에 대한 대략적인 레이아웃 구성입니다. 가장 바깥쪽에는 QHBoxLayout 이 위치합니다. QHBoxLayout 내부에는 두 개의 QVBoxLayout 이 존재하는데 하나는 윈도우의 왼쪽 부분에, 나머지 하나는 윈도우의 오른쪽 부분에 자동으로 배치됩니다. 이처럼 레이아웃 매니저는 다른 레이아웃 매니저를 포함할 수도 있으며 이러한 중첩 구조를 이용하면 복잡한 레이아웃도 구성할 수 있습니다.

왼쪽의 QVBoxLayout 에는 QGroupBox 와 QCheckBox 위젯이 추가되고, 오른쪽의 QVBoxLayout 에는 QTableWidget 객체가 추가됩니다. 왼쪽의 QVBoxLayout 에서는 QCheckBox 위젯을 addWidget 메서드를 사용해 직접 추가하는 것이 아니라 QGroupBox 위젯이 QCheckBox 를 포함하는 형태로 구성했습니다.

QGroupBox 내부에는 9 개의 QCheckBox 가 존재하는데, 칼럼 방향으로 배치하기 위해 또 다른 QVBoxLayout (2)를 사용했습니다. 결과적으로 QCheckBox 위젯들은 내부의 QVBoxLayout (2) 안에 존재하고 QGroupBox 는 QVBoxLayout (2)를 포함합니다.

예제 16.17 레이아웃 중첩을 통한 UI 구성(소스코드: book/ch16/20.py)

예제 16.17 은 그림 16.40 에 나온 프로그램의 전체 코드입니다.

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        self.setGeometry(800, 200, 500, 300)

        groupBox = QGroupBox("검색옵션")
        checkBox1 = QCheckBox("상한가")
        checkBox2 = QCheckBox("하한가")
        checkBox3 = QCheckBox("시가총액 상위")
        checkBox4 = QCheckBox("시가총액 하위")
        checkBox5 = QCheckBox("회전율 상위")
        checkBox6 = QCheckBox("대량거래상위")
        checkBox7 = QCheckBox("환산주가상위")
        checkBox8 = QCheckBox("외국인한도소진상위")
        checkBox9 = QCheckBox("투자자별순위")

        tableWidget = QTableWidget(10, 5)
        tableWidget.setHorizontalHeaderLabels(["종목코드",    "종목명",    "현재가",
"등락률", "거래량"])
        tableWidget.resizeColumnsToContents()
        tableWidget.resizeRowsToContents()

        leftInnerLayOut = QVBoxLayout()
        leftInnerLayOut.addWidget(checkBox1)
        leftInnerLayOut.addWidget(checkBox2)
        leftInnerLayOut.addWidget(checkBox3)
        leftInnerLayOut.addWidget(checkBox4)
        leftInnerLayOut.addWidget(checkBox5)
        leftInnerLayOut.addWidget(checkBox6)
        leftInnerLayOut.addWidget(checkBox7)
        leftInnerLayOut.addWidget(checkBox8)
        leftInnerLayOut.addWidget(checkBox9)
        groupBox.setLayout(leftInnerLayOut)

        leftLayOut = QVBoxLayout()
        leftLayOut.addWidget(groupBox)

        rightLayOut = QVBoxLayout()
        rightLayOut.addWidget(tableWidget)
```

```
        layout = QHBoxLayout()
        layout.addLayout(leftLayOut)
        layout.addLayout(rightLayOut)

        self.setLayout(layout)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    mywindow = MyWindow()
    mywindow.show()
    app.exec_()
```

그림 16.40 에서 윈도우 왼쪽에는 QGroupBox 와 QCheckBox 위젯이 위치합니다. 먼저 사용할 위젯을 생성합니다. 앞서 QGroupBox 와 QCheckBox 위젯을 생성했던 것과 달리 크기나 좌표를 설정하지 않습니다.

```
groupBox = QGroupBox("검색옵션")
checkBox1 = QCheckBox("상한가")
checkBox2 = QCheckBox("하한가")
checkBox3 = QCheckBox("시가총액 상위")
checkBox4 = QCheckBox("시가총액 하위")
checkBox5 = QCheckBox("회전율 상위")
checkBox6 = QCheckBox("대량거래상위")
checkBox7 = QCheckBox("환산주가상위")
checkBox8 = QCheckBox("외국인한도소진상위")
checkBox9 = QCheckBox("투자자별순위")
```

QGroupBox 에 여러 개의 QCheckBox 를 추가할 때 단순히 위젯을 추가하는 것이 아니라 먼저 QVBoxLayout 을 사용해 QCheckBox 위젯을 추가합니다.

```
leftInnerLayOut = QVBoxLayout()
leftInnerLayOut.addWidget(checkBox1)
leftInnerLayOut.addWidget(checkBox2)
leftInnerLayOut.addWidget(checkBox3)
leftInnerLayOut.addWidget(checkBox4)
leftInnerLayOut.addWidget(checkBox5)
leftInnerLayOut.addWidget(checkBox6)
leftInnerLayOut.addWidget(checkBox7)
leftInnerLayOut.addWidget(checkBox8)
leftInnerLayOut.addWidget(checkBox9)
```

QVBoxLayout 에는 QCheckBox 위젯이 포함돼 있으므로 QGroupBox 에 QVBoxLayout 객체만 추가하면 QCheckBox 를 QGroupBox 안쪽으로 배치할 수 있습니다.

```
groupBox.setLayout(leftInnerLayOut)
```

QGroupBox 를 윈도우의 왼쪽에 배치하기 위해 다음과 같이 QVBoxLayout 을 사용해 위젯을 추가합니다.

```
leftLayOut = QVBoxLayout()
leftLayOut.addWidget(groupBox)
```

그림 16.40 에서 윈도우의 오른쪽에는 QTableWidget 이 존재합니다. 먼저 QTableWidget 객체를 생성한 후 컬럼의 라벨을 설정합니다.

```
tableWidget = QTableWidget(10, 5)
tableWidget.setHorizontalHeaderLabels(["종목코드",  "종목명",  "현재가",  "등락률",
"거래량"])
tableWidget.resizeColumnsToContents()
tableWidget.resizeRowsToContents()
```

QTableWidget 객체를 윈도우의 오른쪽에 위치시키기 위해 먼저 QVBoxLayout 클래스의 객체를 생성한 후 QTableWidget 객체를 추가합니다.

```
rightLayOut = QVBoxLayout()
rightLayOut.addWidget(tableWidget)
```

마지막으로 leftLayout 과 rightLayout 을 윈도우의 왼쪽과 오른쪽에 배치하기 위해 QHBoxLayout 클래스의 객체를 생성한 후 레이아웃을 추가합니다.

```
layout = QHBoxLayout()
layout.addLayout(leftLayOut)
layout.addLayout(rightLayOut)
```

이것으로 UI 구성을 모두 마쳤습니다. 현재 윈도우에 레이아웃을 layout 객체로 설정합니다.

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

https://wikidocs.net/5220

5. Dialog

다이얼로그(Dialog)는 '대화'라는 뜻을 가진 영어 단어입니다. GUI 프로그래밍에서 다이얼로그는 사용자와의 상호작용을 위해 사용되는 윈도우를 의미합니다. 예를 들어, 그림 16.42 는 윈도우와 로그인에 사용되는 다이얼로그입니다.

사용자가 메인 윈도우에서 [Sign In] 버튼을 클릭하면 로그인을 위한 새로운 다이얼로그가 생성됩니다. 사용자가 다이얼로그 윈도우에서 아이디와 패스워드를 입력하면 해당 정보가 메인 윈도우에 출력됩니다.

대부분의 GUI 프로그램은 그림 16.42 와 같이 메인 윈도우와 다수의 다이얼로그로 구성됩니다. PyQt 는 다양한 형태의 Dialog 클래스를 지원하므로 여러분은 용도에 맞는 다이얼로그 클래스를 사용하거나 상속을 받은 후 새로운 클래스를 정의해서 사용하면 됩니다.

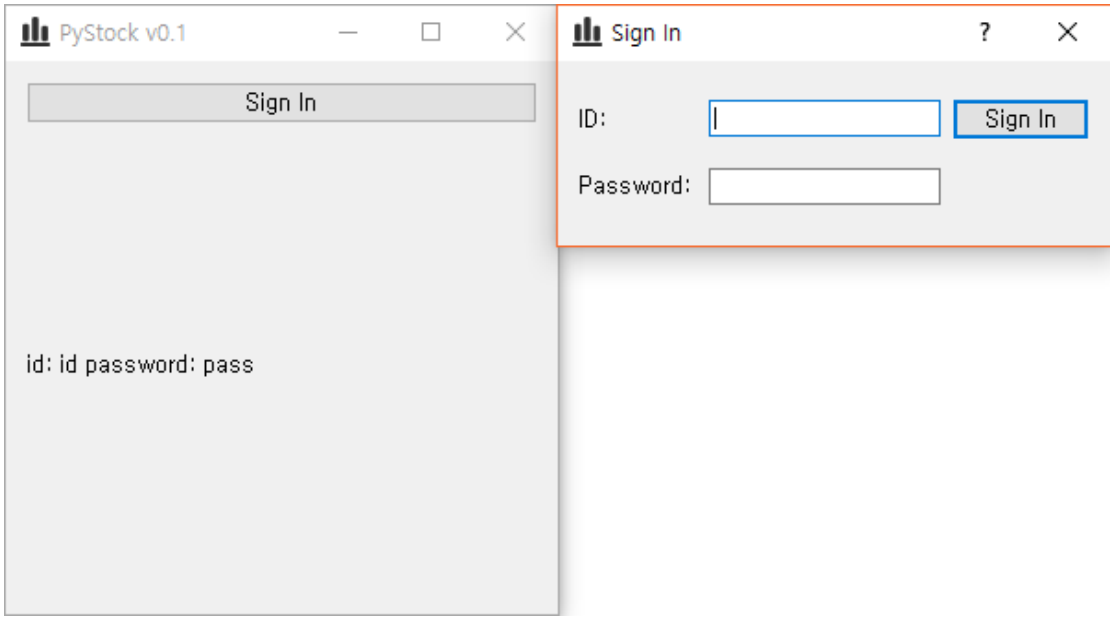


그림 16.42 윈도우와 다이얼로그

1) QFileDialog

이번 절에서는 PyQt 가 제공하는 QFileDigalog 클래스에 대해 알아보겠습니다. QFileDigalog 는 그림 16.43 과 같이 사용자가 파일이나 디렉터리를 선택할 수 있게 하는 다이얼로그 창입니다. 이러한 기능의 창은 자주 사용되기 때문에 PyQt 는 QFileDigalog 클래스를 통해 이러한 기능을 지원하고 있습니다.

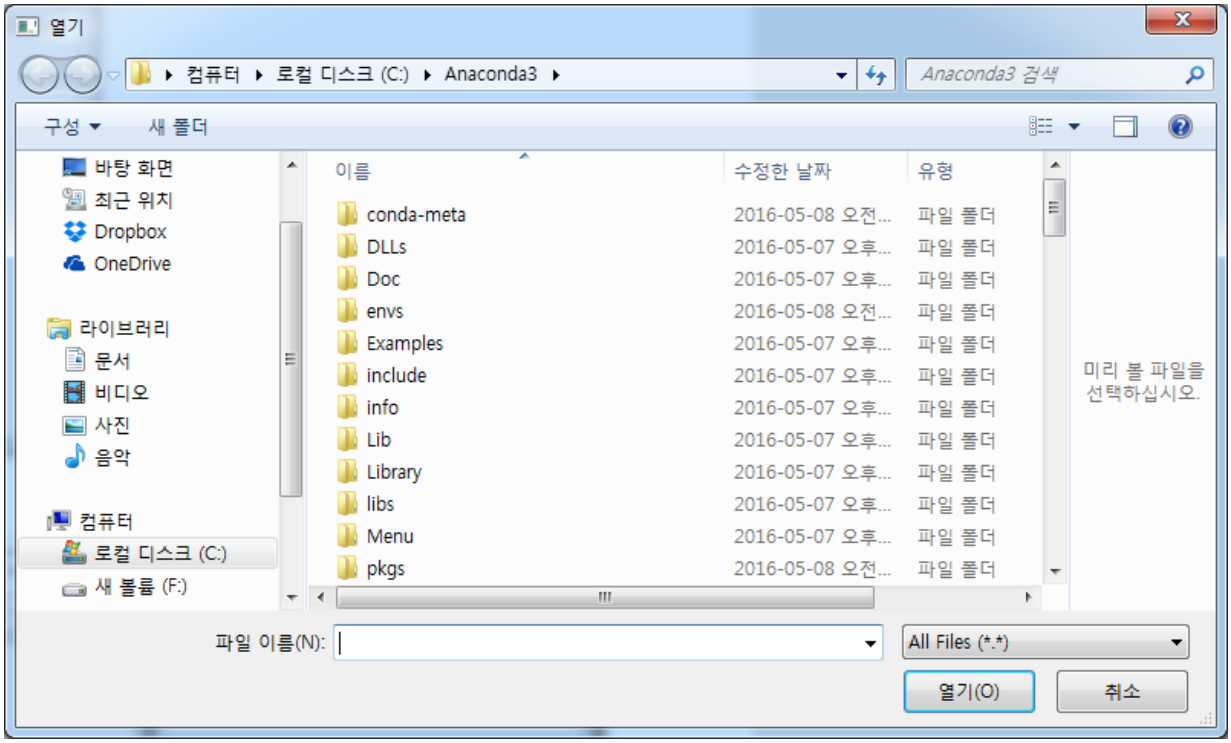


그림 16.43 QFileDialog

그림 16.44 는 QFileDialog 를 사용한 프로그램입니다. 사용자가 [File Open] 버튼을 클릭하면 그림 16.43 의 QFileDialog 창이 화면에 나타납니다. 사용자가 QFileDialog 창을 통해 파일을 선택하면 해당 파일의 절대 경로가 그림 16.44 와 같이 메인 윈도우에 출력됩니다.

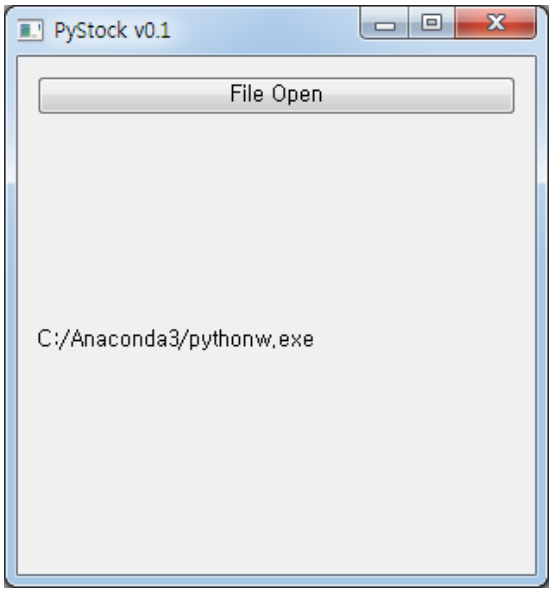


그림 16.44 QFileDialog 를 사용한 응용 프로그램

예제 16.18 QFileDialog 사용 예(소스코드: book/ch16/21.py)

예제 16.18 은 그림 16.44 에 나온 프로그램의 전체 코드입니다.

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QWidget):
    def __init__(self):
```

```
        super().__init__()
        self.setupUI()

    def setupUI(self):
        self.setGeometry(800, 200, 300, 300)
        self.setWindowTitle("PyStock v0.1")

        self.pushButton = QPushButton("File Open")
        self.pushButton.clicked.connect(self.pushButtonClicked)
        self.label = QLabel()

        layout = QVBoxLayout()
        layout.addWidget(self.pushButton)
        layout.addWidget(self.label)

        self.setLayout(layout)

    def pushButtonClicked(self):
        fname = QFileDialog.getOpenFileName(self)
        self.label.setText(fname[0])

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MyWindow()
    window.show()
    app.exec_()
```

그림 16.44 에서 버튼을 클릭하면 pushButtonClicked 라는 슬롯이 호출됩니다. pushButtonClicked 구현을 살펴보면 QFileDialog 클래스의 getOpenFileName 메서드를 호출합니다. getOpenFileName 메서드가 호출되면 그림 16.43 과 같은 QFileDialog 창이 화면에 출력되고, 사용자가 파일을 선택한 후 [열기] 버튼을 클릭하면 해당 파일의 절대 경로가 반환되어 fname 에 저장됩니다.

```
def pushButtonClicked(self):
fname = QFileDialog.getOpenFileName(self)
```

반환된 파일 경로는 튜플 타입으로 fname 이라는 변수가 바인딩하고 있습니다. self.label 객체의 setText 메서드를 통해 튜플의 0 번째 값을 화면에 출력합니다.

```
self.label.setText(fname[0])
```

2) QInputDialog

QInputDialog 는 그림 16.45 와 같이 사용자로부터 간단한 텍스트, 정수, 실수를 받을 때 사용하는 다이얼로그입니다. QInputDialog 의 구성을 살펴보면 상단에는 QLabel 객체가 있고 다음으로 사용자로부터 입력을 받기 위한 QLineEdit 객체가 존재합니다. 마지막으로 OK 와 Cancel 버튼이 있습니다.

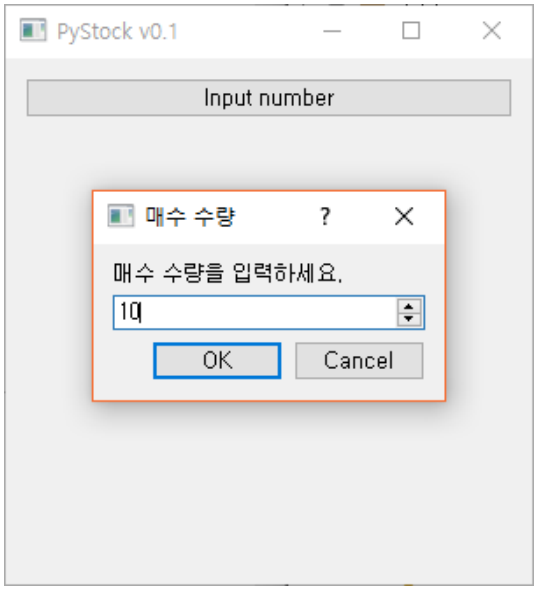


그림 16.45 QInputDialog 를 사용하는 프로그램

예제 16.19 QInputDialog 사용 예(소스코드: book/ch16/22.py)

예제 16.19 는 그림 16.45 에 나온 프로그램의 전체 코드입니다.

```
import sys
from PyQt5.QtWidgets import *

class MyWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        self.setGeometry(800, 200, 300, 300)
        self.setWindowTitle("PyStock v0.1")

        self.pushButton = QPushButton("Input number")
        self.pushButton.clicked.connect(self.pushButtonClicked)
        self.label = QLabel()

        layout = QVBoxLayout()
        layout.addWidget(self.pushButton)
        layout.addWidget(self.label)

        self.setLayout(layout)

    def pushButtonClicked(self):
        text, ok = QInputDialog.getInt(self, '매수 수량', '매수 수량을 입력하세요.')
        if ok:
            self.label.setText(str(text))

if __name__ == "__main__":
    app = QApplication(sys.argv)
```

```
    window = MyWindow()
    window.show()
    app.exec_()
```

예제 16.19 에서 pushButtonClicked 메서드를 살펴보면 QInputDialog 클래스의 getInt 메서드를 호출합니다. getInt 메서드의 첫 번째 인자는 부모 위젯을 의미하고, 두 번째 인자는 QInputDialog 창에 표시될 텍스트입니다. 세 번째 인자는 QInputDialog 창 내부에 출력될 텍스트입니다.

getInt 메서드는 (text, ok)라는 튜플 형태로 값을 반환합니다. text 에는 사용자가 입력한 값이 반환되고, ok 는 사용자가 OK 버튼을 누른 경우에 True 값을 가집니다.

```
def pushButtonClicked(self):
    text, ok = QInputDialog.getInt(self, '매수 수량', '매수 수량을 입력하세요.')
```

사용자가 QInputDialog 에서 OK 버튼을 입력했다면 사용자로부터 입력받은 값을 먼저 문자열로 변경한 후 self.label 객체에 출력합니다.

```
if ok:
    self.label.setText(str(text))
```

QInputDialog 클래스는 getDouble, getText, getItem 이라는 메서드도 갖고 있습니다. 사용자로부터 실수를 입력받을 때는 getInt 대신 getDouble 을 사용하고, 문자열을 입력받을 때는 getText 를 사용합니다.

사용자로부터 단순한 텍스트, 정수, 실수를 입력받는 것이 아니라 그림 16.46 과 같이 다양한 옵션 중 하나를 선택받고자 한다면 getItem 메서드를 사용합니다.

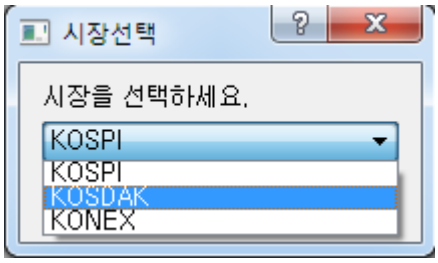


그림 16.46 QInputDialog.getItem() 메서드를 이용한 다이얼로그

getItem 메서드를 사용하는 경우 pushButtonClicked 메서드를 다음과 같이 구현합니다. getItem 메서드의 첫 번째 인자는 부모 위젯, 두 번째는 다이얼로그의 타이틀 텍스트, 세 번째는 다이얼로그 내부의 텍스트, 네 번째 인자는 선택할 아이템 리스트, 다섯 번째 인자는 초기 아이템 인덱스, 여섯 번째 인자는 아이템 수정 가능 여부입니다.

```
def pushButtonClicked(self):
    items = ("KOSPI", "KOSDAK", "KONEX")
    item, ok = QInputDialog.getItem(self, "시장선택", "시장을 선택하세요.", items, 0,
False)
    if ok and item:
        self.label.setText(item)
```

3) 메인 윈도우와 다이얼로그의 상호 작용

이번 절에서는 그림 16.42 의 프로그램을 만들어 보겠습니다. 그림 16.42 는 한 개의 메인 윈도우와 다이얼로그로 구성됩니다. 사용자가 다이얼로그에서 정보를 입력하면 해당 정보가 메인 윈도우 내의 위젯에 출력됩니다.

예제 16.20 메인 윈도우와 다이얼로그의 상호작용(소스코드: book/ch16/24.py)

예제 16.20 은 그림 16.42 에 나온 프로그램의 전체 코드입니다. 먼저 코드를 실행한 후 프로그램의 동작 방식을 파악해 보기 바랍니다. 아이콘 파일이 없는 경우에는 setWindowIcon 메서드를 호출하는 코드를 주석으로 처리하면 됩니다.

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *

class LogInDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.setupUI()

        self.id = None
        self.password = None

    def setupUI(self):
        self.setGeometry(1100, 200, 300, 100)
        self.setWindowTitle("Sign In")
        self.setWindowIcon(QIcon('icon.png'))

        label1 = QLabel("ID: ")
        label2 = QLabel("Password: ")

        self.lineEdit1 = QLineEdit()
        self.lineEdit2 = QLineEdit()
        self.pushButton1= QPushButton("Sign In")
        self.pushButton1.clicked.connect(self.pushButtonClicked)

        layout = QGridLayout()
        layout.addWidget(label1, 0, 0)
        layout.addWidget(self.lineEdit1, 0, 1)
        layout.addWidget(self.pushButton1, 0, 2)
        layout.addWidget(label2, 1, 0)
        layout.addWidget(self.lineEdit2, 1, 1)

        self.setLayout(layout)

    def pushButtonClicked(self):
        self.id = self.lineEdit1.text()
        self.password = self.lineEdit2.text()
        self.close()

class MyWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setupUI()

    def setupUI(self):
        self.setGeometry(800, 200, 300, 300)
        self.setWindowTitle("PyStock v0.1")
        self.setWindowIcon(QIcon('icon.png'))

        self.pushButton = QPushButton("Sign In")
        self.pushButton.clicked.connect(self.pushButtonClicked)
```

```
        self.label = QLabel()

        layout = QVBoxLayout()
        layout.addWidget(self.pushButton)
        layout.addWidget(self.label)

        self.setLayout(layout)

    def pushButtonClicked(self):
        dlg = LogInDialog()
        dlg.exec_()
        id = dlg.id
        password = dlg.password
        self.label.setText("id: %s password: %s" % (id, password))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MyWindow()
    window.show()
    app.exec_()
```

예제 16.20 에는 두 개의 클래스가 정의돼 있습니다. MyWindow 클래스는 QWidget 클래스를 상속받아 정의한 클래스로 지금까지 많이 사용했습니다. 이와 달리 LogInDialog 클래스는 QDialog 클래스를 상속받아 새로 정의한 클래스입니다.

LogInDialog 클래스에서 내부 위젯의 구성은 지금까지 설명한 내용과 동일합니다. 내부 위젯을 배치하는 데 QGridLayout 을 사용했습니다. 지금까지 설명하지 않은 내용으로는 다이얼로그 창에 아이콘을 설정하는 부분입니다. 아이콘 파일은 인터넷을 통해 png 형태의 파일을 내려받은 후 현재 소스코드와 같은 디렉터리에 icon.png 라는 이름으로 저장하면 됩니다. 저자는 https://www.iconfinder.com/에서 아이콘을 내려받았습니다.

```
self.setWindowIcon(QIcon('icon.png'))
```

다이얼로그 창에서 사용자가 아이디와 비밀번호를 입력한 후 [Sign In] 버튼을 누르면 pushButton Clicked 메서드가 호출됩니다. 이 메서드에서는 self.lineEdit1 과 self.lineEdit2 위젯에서 입력된 텍스트를 받아온 후 인스턴스 변수인 self.id, self.password 에 저장합니다. 이처럼 위젯에서 입력받은 값을 인스턴스 변수에 저장해둠으로써 메인 윈도우에서 해당 값들을 읽어갈 수 있게 해줍니다. 마지막으로 close 메서드를 호출해 다이얼로그 창을 닫습니다.

```
def pushButtonClicked(self):
    self.id = self.lineEdit1.text()
    self.password = self.lineEdit2.text()
    self.close()
```

로그인 다이얼로그에 대해 알아봤으니 이번에는 메인 윈도우 역할을 수행하는 MyWindow 클래스에 대해 살펴보겠습니다. MyWindow 클래스 역시 대부분은 지금까지 사용한 MyWindow 클래스와 동일합니다. 다만 메인 윈도우의 버튼 위젯이 클릭될 때 호출되는 pushButtonClicked 메서드에 차이가 있습니다.

pushButtonClicked 메서드에서는 앞서 정의한 LogInDialog 클래스에 대한 인스턴스를 생성합니다. 그리고 인스턴스의 메서드인 exec_를 호출합니다. exec_ 메서드는 다이얼로그 창을 Modal 형태로 출력합니다. 참고로 Modal 다이얼로그는 해당 다이얼로그 창을 닫을 때까지 부모 윈도우로 이동할 수 없습니다.

사용자가 로그인 다이얼로그 창을 닫으면 dlg.exec_() 이후의 구문이 실행됩니다. 앞서 다이얼로그 창에서 인스턴스 변수인 dlg.id 와 dlg.password 에 위젯을 통해 입력받은 값을 저장해뒀으므로 해당 변수를 통해 아이디와

비밀번호를 메인 윈도우로 얻어올 수 있습니다. 메인 윈도우로 얻어온 아이디와 비밀번호를 메인 윈도우의 self.label 객체를 통해 화면에 출력합니다.

```
def pushButtonClicked(self):
    dlg = LogInDialog()
    dlg.exec_()
    id = dlg.id
    password = dlg.password
    self.label.setText("id: %s password: %s" % (id, password))
```

Tello Control Program

- Tello 다운로드 : Documents & Manuals
  https://www.ryzerobotics.com/kr/tello/downloads
  https://github.com/pidster/ryze-tello

- ryze-tello
  https://dl-cdn.ryzerobotics.com/downloads/tello/0228/Tello+SDK+Readme.pdf

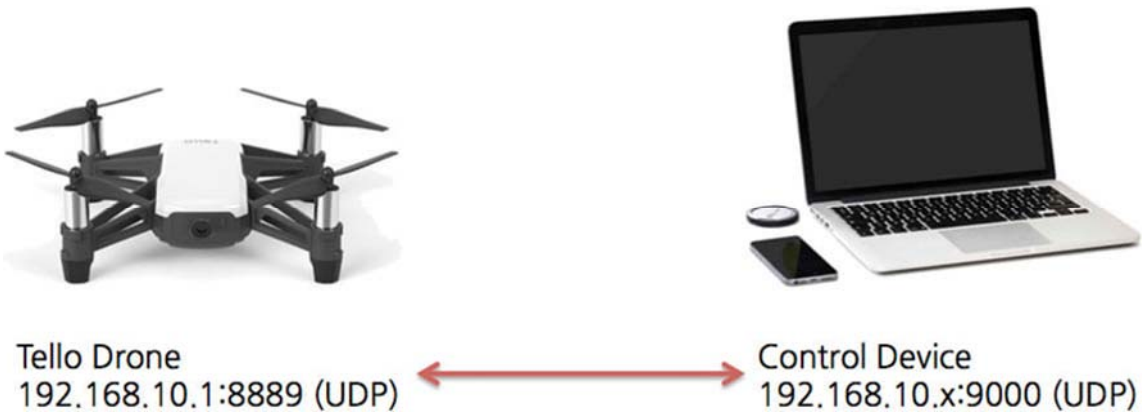
The Tello SDK connects to the aircraft through a Wi-Fi UDP port, allowing users to control the drone with text commands.
Download Tello3.py from
  https://dl-cdn.ryzerobotics.com/downloads/tello/20180222/Tello3.py

◯ Tello package
- easytello 0.0.7 : pip install easytello
  https://pypi.org/project/easytello/
- Tellopy
  https://github.com/hanyazou/TelloPy

---

1. Tello 접속

Tello 의 경우, Tello 가 제공하는 WiFi 네트워크에 접속해서 소켓을 열고 UDP 패킷에 명령어를 전송하는 방식이다.
[출처] https://hooni.net/90225




```
$ ping 192.168.10.1
PING 192.168.10.1 (192.168.10.1) 56(84) bytes of data.
64 bytes from 192.168.10.1: icmp_seq=1 ttl=255 time=1.17 ms
64 bytes from 192.168.10.1: icmp_seq=2 ttl=255 time=1.37 ms
64 bytes from 192.168.10.1: icmp_seq=3 ttl=255 time=4.42 ms
64 bytes from 192.168.10.1: icmp_seq=4 ttl=255 time=1.22 ms
```

◯ Tello 전원을 켜고, PC 에서 Tello Wifi (TELLO-XXXX)에 접속
- Tello Wifi 검색이 안 될때 (펌웨어 업데이트 과정에서 간혹 발생)
  기체가 켜진 상태에서, 전원 버튼을 8 초간 길게 누르면 전원이 꺼짐.
  기체를 다시 켜고, Wifi 검색하면 TELLO-XXXX 주소가 검색 가능

==========

[메모] https://mypages.valdosta.edu/lichen/Python3/DJI.htm

Architecture

- Tello IP: 192.168.10.1
- UDP Port: 8889
- Send Command & Receive Response
- Set up UDP client on PC
- Send "command" to the Tello via UDP Port 8889
- Receive Tello State
- PC Server: 0.0.0.0, Port: 8890
- Receive Tello Video Stream
- PC Server: 0.0.0.0, Port: 11111
- Send "streamon" to the Tello

Commands

- Control Commands
- Set Commands
- Read Commands

Control Commands

```
import socket
import time
tello = ("192.168.10.1", 8889)

def init_drone():
	# create upd client on PC
	try:
		s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
	except socket.error as err:
		print(err)
		exit()
	try:
		# send control commands to the drone
		s.sendto('command', tello)
		time.sleep(5)
	except socket.error as err:
		print(err)
	return s

def takeoff(s):
	s.sendto('takeoff', tello)
	time.sleep(10)

def rotate90(s):
	s.sendto('cw 90', tello)
	time.sleep(10)

def flip_left(s):
	s.sendto('flip l', tello)
	time.sleep(10)

def land(s):
	s.sendto('land', tello)
	time.sleep(5)

def main():
	s = init_drone()
	takeoff(s)
	rotate90(s)
	flip_left(s)
	land(s)

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

Read Commands

```
import socket
import time
tello = ("192.168.10.1", 8889)

def init_drone():
    # create upd client on PC
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    except socket.error as err:
        print(err)
        exit()
    try:
        # send control commands to the drone
        s.sendto('command', tello)
        time.sleep(5)
    except socket.error as err:
        print(err)
    return s

def info(s):
    speed = s.sendto('speed?', tello)
    battery = s.sendto('battery?', tello)
    time = s.sendto('time?', tello)
    height = s.sendto('height?', tello)
    print('----------Informaiton----------')
    print('Speed: %f cm/s' % float(speed))
    print('Battery: %f %%' % float(battery))
    print('Time: %f' % float(time))
    print('Height: %f' % float(height))

def takeoff(s):
    s.sendto('takeoff', tello)
    time.sleep(10)

def land(s):
    s.sendto('land', tello)
    time.sleep(5)

def main():
    s = init_drone()
    takeoff(s)
    info(s)
    land(s)

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

==========

2. Tello 이착륙하는 간단한 Python 코드

- ip, port = (192.168.10.1, 8889)
- command → 5 초 뒤 이륙 (takeoff) → 5 초 뒤 착륙 (land)

```
#!/usr/bin/env python
import socket
import time
tello_ip = '192.168.10.1'
tello_port = 8889
#udp 소켓 생성
socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
tello_address = (tello_ip , tello_port)
# 명령 모드를 사용하기 위해 'command'라는 텍스트를 Tello 에 보냄
socket.sendto('command'.encode('utf-8'),tello_address)
# 5 초간 대기
```

```
time.sleep(5)
# 이륙 'takeoff'라는 텍스트를 Tello 에 보냄
socket.sendto('takeoff'.encode('utf-8'),tello_address)
# 5 초간 대기
time.sleep(5)
# 착륙 'land'라는 텍스트를 Tello 에 보냄
socket.sendto('land'.encode('utf-8'),tello_address)
```

[Note]

- python socket 아래 주소 참조
  https://docs.python.org/ko/3.7/library/socket.html#creating-sockets

- socket.socket(*family*=AF_INET, *type*=SOCK_STREAM, *proto*=0, *fileno*=None) : 지정된 주소 패밀리, 소켓 유형, 및 프로토콜 번호를 사용하여 소켓 객체 생성함. 주소 패밀리는 AF_INET (기본값), AF_INET6, AF_UNIX, AF_CAN, AF_PACKET 또는 AF_RDS 여야 한다. 소켓 유형은 SOCK_STREAM (기본값), SOCK_DGRAM, SOCK_RAW 또는 기타 SOCK_ 상수 중 하나여야 한다. 프로토콜 번호는 일반적으로 0 이며 생략될 수도 있고, 주소 패밀리가 AF_CAN 일 때 프로토콜은 CAN_RAW, CAN_BCM 또는 CAN_ISOTP 중 하나여야 한다.
  - 위의 예제 코드 family=AF_INET (IP4v), type(소켓 타입)=socket.SOCK_DGRAM 데이터그램 소켓
  - family
    AF_INET : IP4v 인터넷 프로토콜
    AF_INET6 : IP6v 인터넷 프로토콜
  - type
    SOCK_STREAM : 연결 지향형 소켓
    Error 나 데이터 Loss 없이 무사히 전달, 전송하는 순서대로 데이터가 전달, 전송되는 데이터의 경계가 존재하지 않음 → 신뢰성있는 순차적인 바이트 기반의 연결 지향 전송 타입
    SOCK_DGRAM : 비연결 지향형 소켓
    전송되는 순서에 상관없이 가장 빠른 전송을 지향, 전송되는 데이터는 손실될 수도 있고, 에러가 발생할 수 있음. 전송되는 데이터의 경계가 존재, 한번에 전송되는 데이터의 크기는 제한

- socket.sendto(*bytes*, *flags*, *address*) : 소켓에 데이터를 보냄. Target 소켓이 address 로 지정되므로, 소켓은 원격 소켓에 연결되지 않아야 한다. 선택적 flags 인자는 recv()와 같은 의미로 전송된 바이트 수를 반환합니다. (address 의 형식은 주소 패밀리에 따라 다르다.)

3. 텍스트 기반 Tello 조정

◯ 데모파일 : Tello3.py

```
#
# Tello Python3 Control Demo
#
# http://www.ryzerobotics.com/
#
# 1/1/2018
import threading
import socket
import sys
import time
host = ''
port = 9000
locaddr = (host,port)
# Create a UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
tello_address = ('192.168.10.1', 8889)
sock.bind(locaddr)

def recv():
    count = 0
```

```
    while True:
        try:
            data, server = sock.recvfrom(1518)
            print(data.decode(encoding="utf-8"))
        except Exception:
            print ('\nExit . . .\n')
            break

print ('\r\n\r\nTello Python3 Demo.\r\n')
print ('Tello: command takeoff land flip forward back left right \r\n up down cw ccw speed speed?\r\n')
print ('end -- quit demo.\r\n')

#recvThread create
recvThread = threading.Thread(target=recv)
recvThread.start()

while True:
    try:
        msg = input("");
        if not msg:
            break
        if 'end' in msg:
            print ('...')
            sock.close()
            break
        # Send data
        msg = msg.encode(encoding="utf-8")
        sent = sock.sendto(msg, tello_address)
    except KeyboardInterrupt:
        print ('\n . . .\n')
        sock.close()
        break
```

[결과]

```
/home/sky/PyCharm/TelloProject/venv/bin/python /home/sky/PyCharm/TelloProject/Tello3.py

Tello Python3 Demo.

Tello: command takeoff land flip forward back left right
    up down cw ccw speed speed?

end -- quit demo.

command
takeoff
up 50
up 50
left 100
right 100
flip r
ccw 10
end
...

Process finished with exit code 0
```

- utf-8 로 부호화된 메시지 msg 를 tello 로 전송함. (sent = sock.sendto(msg, tello_address)).
- command 메시지 보내어 tello 을 명령 모드로 전환한 다음, 다양한 명령(takeoff, land, flip, forward, back, left, right, up, down, cw, ccw, speed 등) 메시지 보내어 tello 을 조정함.
- 종료시 end 메시지 전송

[Note]

- recv() 함수 : 소켓으로부터 데이터를 수신하여 이를 print 출력함. (data,server) =(수신된 데이터, 소켓 주소)

- socket.bind(*address*) : 소켓에 address(또는 protocol, port)를 할당한다. 소켓은 이미 할당되어 있으면 안 된다.
- socket.recvfrom(*bufsize*[, *flags*]) : 소켓에서 데이터를 수신한다. 반환 값은 (bytes, address) 쌍이다. 여기서 bytes 는 수신한 데이터를 나타내는 바이트열 객체이고, address 는 데이터를 보내는 소켓의 주소이다.

- Thread : 파이썬 프로그램은 기본적으로 하나의 쓰레드(Single Thread)에서 실행된다. 즉, 하나의 메인 쓰레드가 파이썬 코드를 순차적으로 실행한다. 코드를 병렬로 실행하기 위해서는 별도의 쓰레드(Subthread)를 생성해야 하는데, 파이썬에서 쓰레드를 생성하기 위해서는 threading 모듈 (High 레벨) 혹은 thread 모듈 (Low 레벨)을 사용할 수 있다. 일반적으로 쓰레드 처리를 위해서는 thread 모듈 위에서 구현된 threading 모듈을 사용하고 있으며, thread 모듈은 (deprecate 되어) 거의 사용하고 있지 않다.

- threading 모듈은 OS 스레드를 객체화한 threaing.Thread 클래스를 제공한다. 기본적으로 스레드를 통해서 실행하는 작업은 파이썬 함수로 작성하게 되는데, Thread 인스턴스를 만들 때 함수와 필요한 인자를 각각 넘겨준 다음 스레드 객체의 .start()를 호출하여 스레드를 시작할 수 있다.

- `recvThread = threading.Thread(target=recv)` : recv 함수를 받은 쓰레드 인스턴스 생성
- `recvThread.start()` : 쓰레드 시작

- threaing.Thread(`name=, target=, args=, kargs=, *, daemon=`) : 쓰레드 객체 생성
  - name : 쓰레드의 이름. Logging 등을 위한 용도로 쓰며 주지 않아도 무방하다.
  - target : 쓰레드에서 실행할 함수
  - args : target 에 넘겨질 인자, 튜플 형식이다.
  - kargs : target 이 키워드 인자를 받을 때 사전으로 넘겨준다.
  - daemon : 데몬 실행 여부. 데몬으로 실행되는 스레드는 프로세스가 종료될 때 즉각 중단된다.

TU Drone

1) 드론 환경 설정하기
   1. 핸드폰에서 Tello 엡을 설치 합니다.
   2. Tello 드론의 벳터리를 분리하고 WIFI ID 를 확인 합니다.
   3. Tello 드론 전원을 켭니다.
   4. 핸드폰에서 Tello ID 를 찾아서 WiFi 로 연결합니다.
   5. 드론을 엑티베이션 합니다.
   6. 드론 베터리를 분리 합니다.

2) PyQt5 된 프로그램을 실행하기 위해 pyqt5 설치
   1. pyqt5 를 사용하므로 python 에 pyqt5 패키지가 설치되어 있어야 합니다.
      $ sudo apt-get install python3-pyqt5
   2. Tello_Gui_E_ver.py 를 python3 로 실행시키면 됩니다.
      $ python3 Tello_Gui_H_ver.py

3) 환경 설정 및 프로그램 복사

   1. 터미널을 오픈 하고 AI_DRONE 디렉토리를 생성 합니다.
      $ conda deactivate
      $ mkdir AI_DRONE
      $ cd AI_DRONE
   2. SW download
      $ git clone http://github.com/AINukeHere/DroneFestival_HS

3. 왼쪽의 파일아이콘을 클릭하고 AI_DRONE DroneFestival_HS 로 이동하고 압축 파일을 클릭하여 풀어줍니다.

4. 터미널에서 DroneFestival_HS 디렉토리로 이동하고 프로그램을 실행합니다.

$ cd AI_DRONE/DroneFestival_HS

$ python3 Tello_Gui_H_ver.py

4) 추론 python 파일 다운로드

$ git clone https://github.com/ikelee77/inference_python.git

PyQt5 설치

$ sudo apt-get install python3-pyqt5

$ python3 Tello_Gui_H_ver.py

[Error]

```
sky@sky:~/sky/DroneFestival_U-master$ python3 Tello_Gui_U_ver.py
Traceback (most recent call last):
  File "Tello_Gui_U_ver.py", line 9, in <module>
    from PyQt5 import QtCore, QtGui, QtWidgets
ImportError: libdouble-conversion.so.1: cannot open shared object file: No such file or
directory
```

sudo apt install --reinstall libdouble-conversion.so.1

```
sky@sky:~/sky/DroneFestival_U-master$ python3 Tello_Gui_U_ver.py
This application failed to start because it could not find or load the Qt platform
plugin "xcb"
in "".

Available platform plugins are: eglfs, linuxfb, minimal, minimalegl, offscreen, vnc,
xcb.

Reinstalling the application may fix this problem.
중지됨 (core dumped)
```

```
sudo apt-get --reinstall install libqt5dbus5 libqt5widgets5 libqt5network5
libqt5gui5 libqt5core5a libdouble-conversion1 libxcb-xinerama0
```

```
sky@sky:~/sky/DroneFestival_U-master$ python3 Tello_Gui_U_ver.py
usage : python Tello_Gui_U_ver.py [True|False]
```

Pycharm – PyQt5 설치 안 되는 거

해결 : Anaconda 로 해결

1) Anaconda 설치 :

$ sha256sum Anaconda3-2019.10-Linux-x86_64.sh

$ bash Anaconda3-2019.10-Linux-x86_64.sh

$ source ~/.bashrc

2) 가상환경 생성

$ conda create -n 가상환경명 python=3.6

3) 가상환경으로 source ~/.bashrc → pycharm 실행

→ [File]-[Settings]-[Project Interpreter]-[톱니바퀴 설정]-[System Interpreter]-[[Anaconda 설치 디렉토리]→[bin]→python3 설정

[Memo]

#ubuntu 16.04 계열 설치방법입니다.

$ sudo apt install python3-pyqt5
$ sudo apt install pyqt5-dev-tools

# 디자이너 설치
$ sudo apt install qttools5-dev-tools

# 디자이너 실행
$ qtchooser -run-tool=designer -qt=5

혹은 아래 파일을 다음과 같이 수정
/usr/lib/x86_64-linux-gnu/qt-default/qtchooser/default.conf
/usr/lib/x86_64-linux-gnu/qt5/bin
/usr/lib/x86_64-linux-gnu

[출처] PyQt5 설치법 (Ubuntu)|작성자 StillAlive

Qt 사용하기
https://nanite.tistory.com/53

Anaconda, PyQt5, Pycharm 환경으로 실습

4. PyQt5 로 간단한 UI 작성한 Tello 제어

Python 에 의한 Tello 제어

https://midoriit.com/2018/05/python%E3%81%AB%E3%82%88%E3%82%8B%E3%83%89%E3%83%AD%E3%83%BC%E3%83%B3%E3%80%8Ctello%E3%80%8D%E3%81%AE%E5%88%B6%E5%BE%A1.html

1 단계) Version1

TelloSampe / TelloController1.py

간단한 명령어 (command, takeoff, land) + PyQt5
리턴이 TRUE/ FALSE 로 되어 있어도 실제로는 ERROR 가 리턴될 수 있다.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import threading
import socket
import time
import sys
from PyQt5.QtWidgets import *

class TelloController1(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        self.initConnection()
        self.initUI()

    # 통신 설정
    def initConnection(self):
        host = ''
        port = 9000
        locaddr = (host,port)
        self.tello = ('192.168.10.1', 8889)
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind(locaddr)
        # 수신 Thread 시작
        recvThread = threading.Thread(target=self.recvSocket)
        recvThread.setDaemon(True)
        recvThread.start()
        # 먼저 command 명령 전송
        try:
            sent = self.sock.sendto('command'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    # UI 작성
    def initUI(self):
        self.label = QLabel('')
        self.label.setFrameStyle(QFrame.Box | QFrame.Plain)
        # 종료 버튼
        endBtn = QPushButton("End")
        endBtn.clicked.connect(self.endBtnClicked)
        # 이착륙 버튼
        takeoffBtn = QPushButton("Takeoff")
        takeoffBtn.clicked.connect(self.takeoffBtnClicked)
        landBtn = QPushButton("Land")
        landBtn.clicked.connect(self.landBtnClicked)
        # 버튼 레이아웃
        layout = QGridLayout()
        layout.addWidget(self.label,0,0)
        layout.addWidget(endBtn,0,1)
        layout.addWidget(takeoffBtn,1,0)
        layout.addWidget(landBtn,1,1)
```

```
            self.setLayout(layout)
        # 종료
        def endBtnClicked(self):
            sys.exit()
        # takeoff 명령 전송
        def takeoffBtnClicked(self):
            try:
                sent = self.sock.sendto('takeoff'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        # land 명령 전송
        def landBtnClicked(self):
            try:
                sent = self.sock.sendto('land'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        # Tello 의 응답 수신
        def recvSocket(self):
            while True:
                try:
                    data, server = self.sock.recvfrom(1518)
                    self.label.setText(data.decode(encoding="utf-8"))
                except:
                    pass

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = TelloController1()
    window.show()
    sys.exit(app.exec_())
```

[메모 : Python]

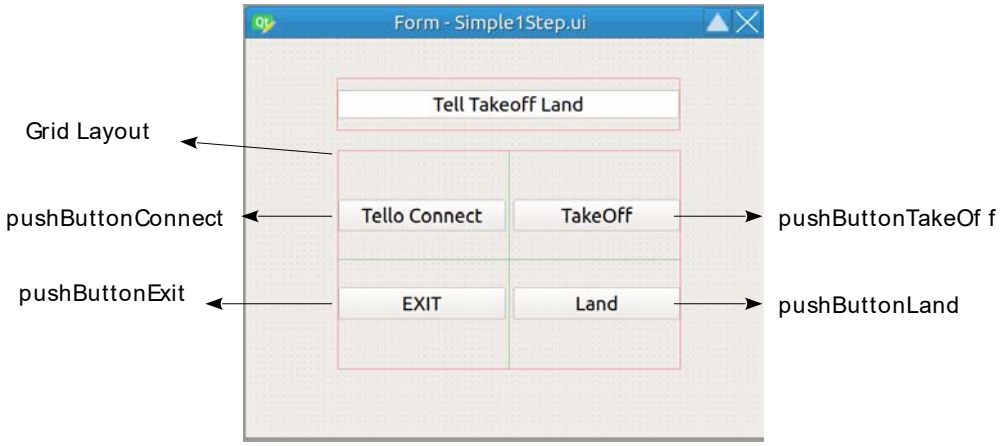
1) `if __name__ == '__main__':`
   - https://dojang.io/mod/page/view.php?id=24480
   - 파이썬은 최초로 시작하는 스크립트 파일과 모듈의 차이가 없습니다. 어떤 스크립트 파일이든 시작점도 될 수 있고, 모듈도 될 수 있습니다. 그래서 __name__ 변수를 통해 현재 스크립트 파일이 시작점인지 모듈인지 판단합니다.
   - if __name__ == '__main__':처럼 __name__ 변수의 값이 '__main__'인지 확인하는 코드는 현재 스크립트 파일이 프로그램의 시작점이 맞는지 판단하는 작업입니다. 즉, 스크립트 파일이 메인 프로그램으로 사용될 때와 모듈로 사용될 때를 구분하기 위한 용도입니다
   - __name__ : 모듈의 이름이 저장되는 변수이며 import 로 모듈을 가져왔을 때 모듈의 이름이 들어간다. 하지만 파이썬 인터프리터로 스크립트 파일을 직접 실행했을 때는 모듈의 이름이 아니라 '__main__'이 들어간다.

2) sys.argv (인자 전달)
   - 파이썬 스크립트 실행시 인자값을 전달받아 실행하면 편리한 경우가 많다. python 스크립트의 명령행 인자들은 argv 배열에 전달됩니다. argv[0]에는 실행하는 스크립트의 경로가 담겨 있고, argv 배열의 1 번째 원소부터 인자로 전달되는 값들을 저장하고 있습니다.

3) app = QApplication(sys.argv)
   - https://wikidocs.net/5222
   - PyQt 에서는 QApplication 객체에서 exec_ 메서드를 호출해 이벤트 루프를 생성합니다. 한번 생성된 이벤트 루프는 사용자가 윈도우를 닫을 때까지 실행되면서 위젯에서 발생한 시그널을 처리하는 슬롯을 호출하는 역할을 합니다

※ PyQt5 – Qt Designer 로 수정

Simple1Step.ui



TelloController1-UI.py

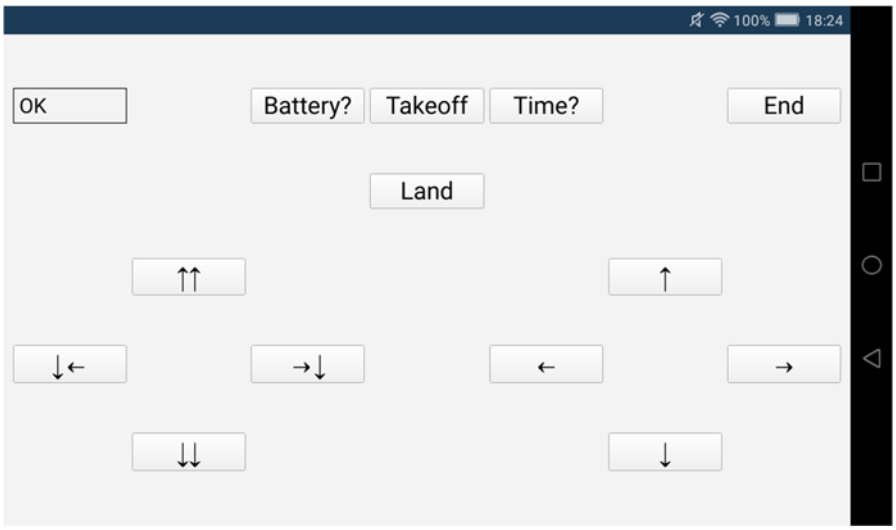
```
import sys
import threading, socket, time
from PyQt5.QtWidgets import *
from PyQt5 import uic
form_class = uic.loadUiType("Simple1Step.ui")[0]
class MyWindow(QMainWindow, form_class):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.pushButtonConnect.clicked.connect(self.btn_clicked_Connect)
        self.pushButtonTakeOff.clicked.connect(self.btn_clicked_TakeOff)
        self.pushButionLand.clicked.connect(self.btn_clicked_Land)
        self.pushButtonExit.clicked.connect(self.btn_clicked_Exit)
    def btn_clicked_Connect(self):
        print("Connect")
        host = ''
        port = 9000
        locaddr = (host,port)
        self.tello = ('192.168.10.1', 8889)
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind(locaddr)
        # 수신 Thread 시작
        recvThread = threading.Thread(target=self.recvSocket)
        recvThread.setDaemon(True)
        recvThread.start()
        # 먼저 command 명령 전송
        try:
            sent = self.sock.sendto('command'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def btn_clicked_TakeOff(self):
        print("TakeOff")
        try:
            sent = self.sock.sendto('takeoff'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def btn_clicked_Land(self):
        print("Land")
        try:
            sent = self.sock.sendto('land'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def btn_clicked_Exit(self):
        print("Exit")
        sys.exit()
    # Tello 의 응답 수신
    def recvSocket(self):
        while True:
            try:
                data, server = self.sock.recvfrom(1518)
```

```
                    self.label.setText(data.decode(encoding="utf-8"))
                except:
                    pass
if __name__ == "__main__":
    app = QApplication(sys.argv)
    myWindow = MyWindow()
    myWindow.show()
    app.exec_()
```

2 단계) Version2

TelloSampe / TelloController2.py

이전 작성한 프로그램을 확장하고 SDK 설명서에 나와있는 거의 모든 명령을 실행합니다. 그러나 명령의 매개 변수는 고정입니다.




```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import threading
import socket
import time
import sys
from PyQt5.QtWidgets import *
class TelloController1(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        self.initConnection()
        self.initUI()
    # 通信の設定
    def initConnection(self):
        host = ''
        port = 9000
        locaddr = (host,port)
        self.tello = ('192.168.10.1', 8889)
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind(locaddr)
        # 受信スレッド起動
        recvThread = threading.Thread(target=self.recvSocket)
        recvThread.setDaemon(True)
        recvThread.start()
        # 最初にcommand コマンドを送信
        try:
            sent = self.sock.sendto('command'.encode(encoding="utf-8"), self.tello)
        except:
```

```
            pass
        # 速度を遅めに設定
        try:
            sent = self.sock.sendto('speed 50'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    # UI の作成
    def initUI(self):
        self.label = QLabel('')
        self.label.setFrameStyle(QFrame.Box | QFrame.Plain)
        # 終了ボタン
        endBtn = QPushButton("End")
        endBtn.clicked.connect(self.endBtnClicked)
        # 離着陸ボタン
        takeoffBtn = QPushButton("Takeoff")
        takeoffBtn.clicked.connect(self.takeoffBtnClicked)
        landBtn = QPushButton("Land")
        landBtn.clicked.connect(self.landBtnClicked)
        # 上昇下降回転ボタン
        upBtn = QPushButton("↑")
        upBtn.clicked.connect(self.upBtnClicked)
        downBtn = QPushButton("↓")
        downBtn.clicked.connect(self.downBtnClicked)
        cwBtn = QPushButton("→")
        cwBtn.clicked.connect(self.cwBtnClicked)
        ccwBtn = QPushButton("←")
        ccwBtn.clicked.connect(self.ccwBtnClicked)
        # 前後左右ボタン
        forwardBtn = QPushButton("↑")
        forwardBtn.clicked.connect(self.forwardBtnClicked)
        backBtn = QPushButton("↓")
        backBtn.clicked.connect(self.backBtnClicked)
        rightBtn = QPushButton("→")
        rightBtn.clicked.connect(self.rightBtnClicked)
        leftBtn = QPushButton("←")
        leftBtn.clicked.connect(self.leftBtnClicked)
        batteryBtn = QPushButton("Battery?")
        batteryBtn.clicked.connect(self.batteryBtnClicked)
        timeBtn = QPushButton("Time?")
        timeBtn.clicked.connect(self.timeBtnClicked)
        # ボタンのレイアウト
        layout = QGridLayout()
        layout.addWidget(self.label,0,0)
        layout.addWidget(endBtn,0,6)
        layout.addWidget(takeoffBtn,0,3)
        layout.addWidget(landBtn,1,3)
        layout.addWidget(upBtn,2,1)
        layout.addWidget(downBtn,4,1)
        layout.addWidget(cwBtn,3,2)
        layout.addWidget(ccwBtn,3,0)
        layout.addWidget(forwardBtn,2,5)
        layout.addWidget(backBtn,4,5)
        layout.addWidget(rightBtn,3,6)
        layout.addWidget(leftBtn,3,4)
        layout.addWidget(batteryBtn,0,2)
        layout.addWidget(timeBtn,0,4)
        self.setLayout(layout)
    # 終了処理
    def endBtnClicked(self):
        sys.exit()
    # 各種コマンド送信
    def takeoffBtnClicked(self):
        try:
            sent = self.sock.sendto('takeoff'.encode(encoding="utf-8"), self.tello)
```

```
            except:
                pass
        def landBtnClicked(self):
            try:
                sent = self.sock.sendto('land'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        def upBtnClicked(self):
            try:
                sent = self.sock.sendto('up 20'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        def downBtnClicked(self):
            try:
                sent = self.sock.sendto('down 20'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        def cwBtnClicked(self):
            try:
                sent = self.sock.sendto('cw 45'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        def ccwBtnClicked(self):
            try:
                sent = self.sock.sendto('ccw 45'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        def forwardBtnClicked(self):
            try:
                sent = self.sock.sendto('forward 20'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        def backBtnClicked(self):
            try:
                sent = self.sock.sendto('back 20'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        def rightBtnClicked(self):
            try:
                sent = self.sock.sendto('right 20'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        def leftBtnClicked(self):
            try:
                sent = self.sock.sendto('left 20'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        def batteryBtnClicked(self):
            try:
                sent = self.sock.sendto('battery?'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        def timeBtnClicked(self):
            try:
                sent = self.sock.sendto('time?'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        # Tello からのレスポンス受信
        def recvSocket(self):
            while True:
                try:
                    data, server = self.sock.recvfrom(1518)
                    self.label.setText(data.decode(encoding="utf-8").strip())
                except:
                    pass
if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = TelloController1()
```

```
    window.show()
    sys.exit(app.exec_())
```

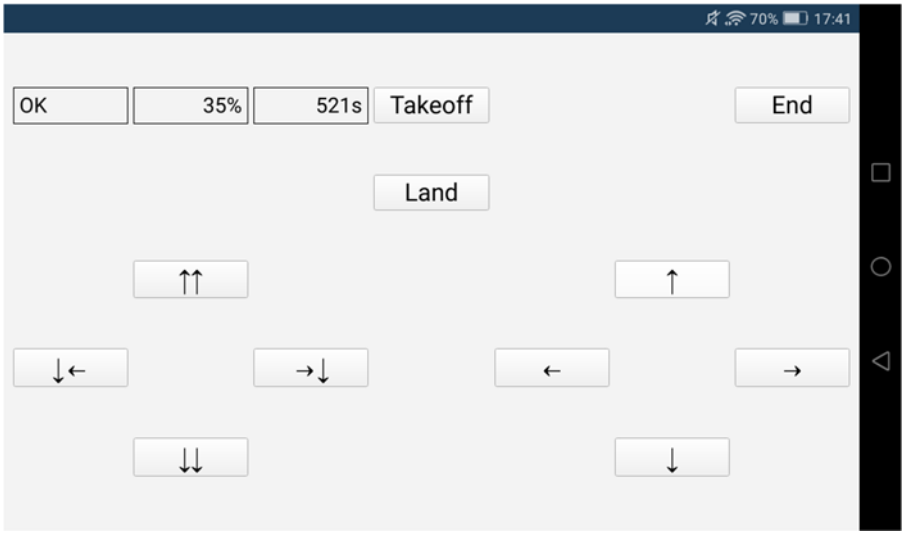
- "command"명령을 전송 한 후에 "speed 50"명령으로 속도를 조금 늦은 50cm / s 로 설정하고 있습니다.
- 단일 명령에 대해 상승 하강과 전후 좌우 이동은 20cm 에 회전은 45 °로 고정되어 있습니다.
- "speed?」나 「battery?"명령은 SDK 설명서에서 첫 글자가 대문자이지만, 소문자 않으면 에러가 되어 버립니다. 또한 이러한 명령의 응답은 마지막에 개행 코드가 붙는 경우가 있기 때문에 strip()에서 제거하고 있습니다.

- 이 프로그램 Tello 를 조종 해보고 안 것은, Tello 가 벽이나 바닥에 너무 가까이있는 것 같은 경우 명령대로 작동하지 않고 error 응답을 반환 할 많다는 것입니다.
- 전용 응용 프로그램을 사용 있어도 벽에 너무 가까이 있기 때문에 떠나려했지만 결국 부딪쳐 버렸다는 것이 지금까지도 몇 번 있었습니다. 그런 경우에도 Tello 이 명령대로 작동 할 수없는 상태에 빠졌던 수 있습니다.

3 단계) Version3

TelloSampe / TelloController3.py

이번에는 배터리의 충전 상태와 비행 시간을 정기적으로 자동으로 검색하고 라벨에 표시하도록합니다. 아래 그림에서 가장 왼쪽의 "OK"보통 명령에 대한 응답 그 오른쪽의 "35 %"는 충전 "521s"비행 시간입니다.



지난번 만든 프로그램을 실행 한 결과 "battery? '에 대해서는 숫자 만 반환"time? "에 대한 응답으로는 숫자 뒤에's '가 붙는 것을 알 수 있었으므로,이 사양을 이용합니다. 그러나 이 사양은 SDK 설명서의 내용과는 다르기 때문에 앞으로 변경 될 가능성도 있습니다.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import threading
import socket
import time
import sys
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
class TelloController1(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        self.initConnection()
        self.initUI()
        # command 명령 전송
        try:
```

```
            sent = self.sock.sendto('command'.encode(encoding="utf-8"), self.tello)
        except:
            pass
        # 속도 낮추는 조절
        try:
            sent = self.sock.sendto('speed 50'.encode(encoding="utf-8"), self.tello)
        except:
            pass
        # ask Thread 시작
        askThread = threading.Thread(target=self.askTello)
        askThread.setDaemon(True)
        askThread.start()
    # 통신 설정
    def initConnection(self):
        host = ''
        port = 9000
        locaddr = (host,port)
        self.tello = ('192.168.10.1', 8889)
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind(locaddr)
        # 受信スレッド起動
        recvThread = threading.Thread(target=self.recvSocket)
        recvThread.setDaemon(True)
        recvThread.start()
    # UI 작성
    def initUI(self):
        # 情報表示用ラベル
        self.label = QLabel('')
        self.label.setFrameStyle(QFrame.Box | QFrame.Plain)
        self.batteryLabel = QLabel('100%')
        self.batteryLabel.setFrameStyle(QFrame.Box | QFrame.Plain)
        self.batteryLabel.setAlignment(Qt.AlignBottom | Qt.AlignRight)
        self.timeLabel = QLabel('0s')
        self.timeLabel.setFrameStyle(QFrame.Box | QFrame.Plain)
        self.timeLabel.setAlignment(Qt.AlignBottom | Qt.AlignRight)
        # 종료 버튼
        endBtn = QPushButton("End")
        endBtn.clicked.connect(self.endBtnClicked)
        # 이착륙 버튼
        takeoffBtn = QPushButton("Takeoff")
        takeoffBtn.clicked.connect(self.takeoffBtnClicked)
        landBtn = QPushButton("Land")
        landBtn.clicked.connect(self.landBtnClicked)
        # 상승, 하강, 회전 버튼
        upBtn = QPushButton("↑↑")
        upBtn.clicked.connect(self.upBtnClicked)
        downBtn = QPushButton("↓↓")
        downBtn.clicked.connect(self.downBtnClicked)
        cwBtn = QPushButton("→↓")
        cwBtn.clicked.connect(self.cwBtnClicked)
        ccwBtn = QPushButton("↓←")
        ccwBtn.clicked.connect(self.ccwBtnClicked)
        # 전후 좌우 버튼
        forwardBtn = QPushButton("↑")
        forwardBtn.clicked.connect(self.forwardBtnClicked)
        backBtn = QPushButton("↓")
        backBtn.clicked.connect(self.backBtnClicked)
        rightBtn = QPushButton("→")
        rightBtn.clicked.connect(self.rightBtnClicked)
        leftBtn = QPushButton("←")
        leftBtn.clicked.connect(self.leftBtnClicked)
        # UI 레이아웃
        layout = QGridLayout()
        layout.addWidget(self.label,0,0)
```

```
        layout.addWidget(self.batteryLabel,0,1)
        layout.addWidget(self.timeLabel,0,2)
        layout.addWidget(endBtn,0,6)
        layout.addWidget(takeoffBtn,0,3)
        layout.addWidget(landBtn,1,3)
        layout.addWidget(upBtn,2,1)
        layout.addWidget(downBtn,4,1)
        layout.addWidget(cwBtn,3,2)
        layout.addWidget(ccwBtn,3,0)
        layout.addWidget(forwardBtn,2,5)
        layout.addWidget(backBtn,4,5)
        layout.addWidget(rightBtn,3,6)
        layout.addWidget(leftBtn,3,4)
        self.setLayout(layout)
    # 종료
    def endBtnClicked(self):
        sys.exit()
    # 각종 명령 전송
    def takeoffBtnClicked(self):
        try:
            sent = self.sock.sendto('takeoff'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def landBtnClicked(self):
        try:
            sent = self.sock.sendto('land'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def upBtnClicked(self):
        try:
            sent = self.sock.sendto('up 20'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def downBtnClicked(self):
        try:
            sent = self.sock.sendto('down 20'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def cwBtnClicked(self):
        try:
            sent = self.sock.sendto('cw 45'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def ccwBtnClicked(self):
        try:
            sent = self.sock.sendto('ccw 45'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def forwardBtnClicked(self):
        try:
            sent = self.sock.sendto('forward 20'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def backBtnClicked(self):
        try:
            sent = self.sock.sendto('back 20'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def rightBtnClicked(self):
        try:
            sent = self.sock.sendto('right 20'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def leftBtnClicked(self):
        try:
            sent = self.sock.sendto('left 20'.encode(encoding="utf-8"), self.tello)
        except:
            pass
```

```
    # Tello 응답 수신
    def recvSocket(self):
        while True:
            try:
                data, server = self.sock.recvfrom(1518)
                resp = data.decode(encoding="utf-8").strip()
                if resp.isdecimal():    # 숫자일 경우 충전량 표시
                    self.batteryLabel.setText(resp + "%")
                elif resp[-1:] == "s":  # 마지막 글자가 's'일 경우, 비행 시간 표시
                    self.timeLabel.setText(resp)
                elif resp == "OK":      # OK 는 Black
                    self.label.setStyleSheet("color:black;")
                    self.label.setText(resp)
                else:                   # 그렇지 않으면 Red
                    self.label.setStyleSheet("color:red;")
                    self.label.setText(resp)
            except:
                pass
    # Ask
    def askTello(self):
        while True:
            try:
                sent = self.sock.sendto('battery?'.encode(encoding="utf-8"), self.tello)
            except:
                pass
            time.sleep(0.5)
            try:
                sent = self.sock.sendto('time?'.encode(encoding="utf-8"), self.tello)
            except:
                pass
            time.sleep(0.5)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = TelloController1()
    window.show()
    sys.exit(app.exec_())
```

- 충전 및 비행 시간을 가져 오는 명령은 스레드에서 0.5 초 간격으로 보냅니다.

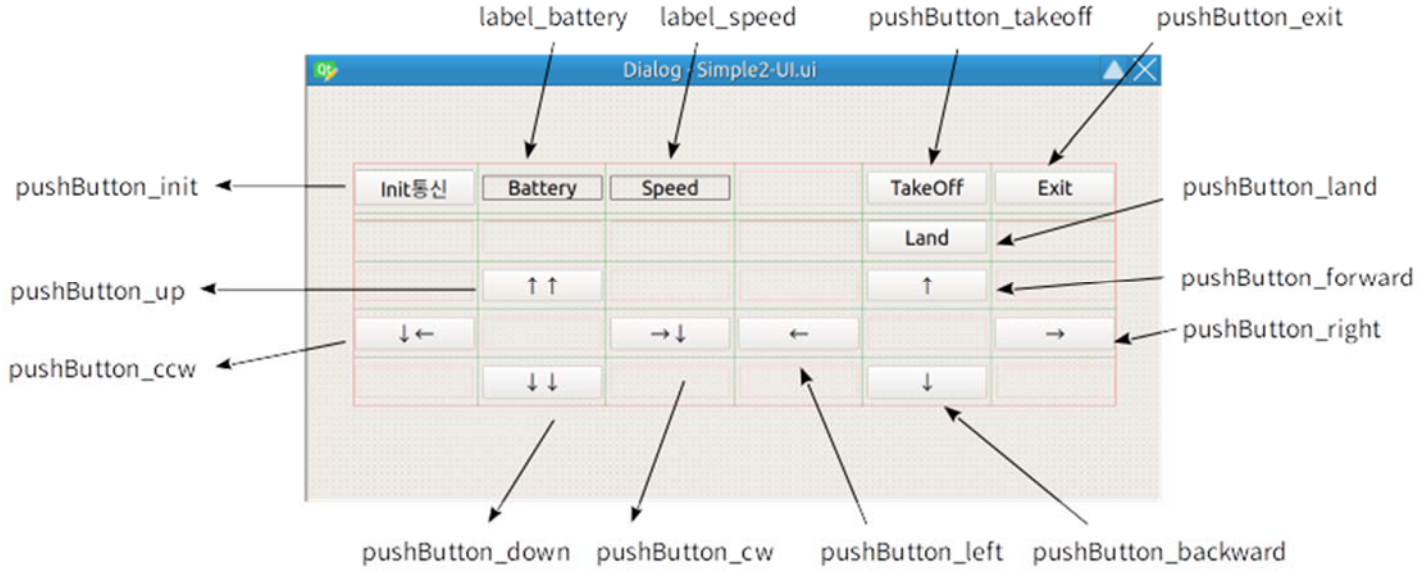
```
    # 問い合わせ
    def askTello(self):
        while True:
            try:
                sent = self.sock.sendto('battery?'.encode(encoding="utf-8"), self.tello)
            except:
                pass
            time.sleep(0.5)
            try:
                sent = self.sock.sendto('time?'.encode(encoding="utf-8"), self.tello)
            except:
                pass
```

- Tello 에서 응답을 수신했을 때 숫자 만 있다면 충전 마지막 문자가 "s"라면 비행 시간이라는 판정을 표시하는 라벨을 선택합니다. 그렇지 않으면에는 "OK"여부에 따라 라벨의 색상을 변경할 수 있습니다.

```
                if resp.isdecimal():    # 数字だけなら充電量
                    self.batteryLabel.setText(resp + "%")
                elif resp[-1:] == "s":  # 最後の文字がs なら飛行時間
                    self.timeLabel.setText(resp)
                elif resp == "OK":      # OK は黒
                    self.label.setStyleSheet("color:black;")
                    self.label.setText(resp)
                else:                   # それ以外は赤
                    self.label.setStyleSheet("color:red;")
                    self.label.setText(resp)
```

※ PyQt5 – Qt Designer 로 수정

Simple2Step.ui




```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import sys
import threading, socket, time
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5 import uic
form_class = uic.loadUiType("Simple2-UI.ui")[0]
class TelloController(QDialog, form_class):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.pushButton_init.clicked.connect(self.btn_clicked_Init)
        self.pushButton_exit.clicked.connect(self.btn_clicked_Exit)
        self.pushButton_takeoff.clicked.connect(self.btn_clicked_TakeOff)
        self.pushButton_land.clicked.connect(self.btn_clicked_Land)
        self.pushButton_up.clicked.connect(self.btn_clicked_Up)
        self.pushButton_down.clicked.connect(self.btn_clicked_Down)
        self.pushButton_cw.clicked.connect(self.btn_clicked_CW)
        self.pushButton_ccw.clicked.connect(self.btn_clicked_CCW)
        self.pushButton_forward.clicked.connect(self.btn_clicked_Forward)
        self.pushButton_backward.clicked.connect(self.btn_clicked_Backward)
        self.pushButton_left.clicked.connect(self.btn_clicked_Left)
        self.pushButton_right.clicked.connect(self.btn_clicked_Right)
        self.label = QLabel('')
        self.label.setFrameStyle(QFrame.Box | QFrame.Plain)
    def btn_clicked_Init(self):
        print("Connect")
        host = ''
        port = 9000
        locaddr = (host,port)
        self.tello = ('192.168.10.1', 8889)
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind(locaddr)
        # 수신 Thread 시작
        recvThread = threading.Thread(target=self.recvSocket)
        recvThread.setDaemon(True)
        recvThread.start()
        # 먼저 command 명령 전송
        try:
            sent = self.sock.sendto('command'.encode(encoding="utf-8"), self.tello)
        except:
```

```
            pass
        # 속도 낮추는 조절
        try:
            sent = self.sock.sendto('speed 50'.encode(encoding="utf-8"), self.tello)
        except:
            pass
        # ask Thread 시작
        askThread = threading.Thread(target=self.askTello)
        askThread.setDaemon(True)
        askThread.start()
    def btn_clicked_Exit(self):
        print("Exit")
        sys.exit()
    def btn_clicked_TakeOff(self):
        print("TakeOff")
        try:
            sent = self.sock.sendto('takeoff'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def btn_clicked_Land(self):
        print("Land")
        try:
            sent = self.sock.sendto('land'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def btn_clicked_Up(self):
        print("Up")
        try:
            sent = self.sock.sendto('up 20'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def btn_clicked_Down(self):
        print("Down")
        try:
            sent = self.sock.sendto('down 20'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def btn_clicked_CW(self):
        print("CW")
        try:
            sent = self.sock.sendto('cw 45'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def btn_clicked_CCW(self):
        print("CCW")
        try:
            sent = self.sock.sendto('ccw 45'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def btn_clicked_Forward(self):
        print("Forward")
        try:
            sent = self.sock.sendto('forward 20'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def btn_clicked_Backward(self):
        print("Backward")
        try:
            sent = self.sock.sendto('back 20'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def btn_clicked_Right(self):
        print("Right")
        try:
            sent = self.sock.sendto('right 20'.encode(encoding="utf-8"), self.tello)
        except:
            pass
    def btn_clicked_Left(self):
```

```
            print("Left")
            try:
                sent = self.sock.sendto('left 20'.encode(encoding="utf-8"), self.tello)
            except:
                pass
        # Tello 응답 수신
        def recvSocket(self):
            while True:
                try:
                    data, server = self.sock.recvfrom(1518)
                    resp = data.decode(encoding="utf-8").strip()
                    if resp.isdecimal():    # 숫자일 경우 충전량 표시
                        self.label_battery.setText(resp + "%")
                    elif resp[-1:] == "s":  # 미자막 글자가 's'일 경우, 비행 시간 표시
                        self.label_speed.setText(resp)
                    elif resp == "OK":      # OK 는 Black
                        self.label.setStyleSheet("color:black;")
                        self.label.setText(resp)
                    else:                   # 그렇지 않으면 Red
                        self.label.setStyleSheet("color:red;")
                        self.label.setText(resp)
                except:
                    pass
        # Ask
        def askTello(self):
            while True:
                try:
                    sent = self.sock.sendto('battery?'.encode(encoding="utf-8"), self.tello)
                except:
                    pass
                time.sleep(0.5)
                try:
                    sent = self.sock.sendto('time?'.encode(encoding="utf-8"), self.tello)
                except:
                    pass
                time.sleep(0.5)
    if __name__ == '__main__':
        app = QApplication(sys.argv)
        window = TelloController()
        window.show()
        sys.exit(app.exec_())
```

[참고 2] Camera Video Using PyQt5

```
import sys
import cv2
import numpy as np
import threading
import time
import queue
from PyQt5.QtWidgets import *
from PyQt5 import QtCore, QtGui, uic
running = False
capture_thread = None
form_class = uic.loadUiType("simple.ui")[0]
q = queue.Queue()
def grab(cam, queue, width, height, fps):
    global running
    capture = cv2.VideoCapture(cam)
    capture.set(cv2.CAP_PROP_FRAME_WIDTH, width)
    capture.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
    capture.set(cv2.CAP_PROP_FPS, fps)
    while (running):
        frame = {}
        capture.grab()
        retval, img = capture.retrieve(0)
        frame["img"] = img
        if queue.qsize() < 10:
            queue.put(frame)
        else:
            print
            queue.qsize()
class OwnImageWidget(QWidget):
    def __init__(self, parent=None):
        super(OwnImageWidget, self).__init__(parent)
        self.image = None
    def setImage(self, image):
        self.image = image
        sz = image.size()
        self.setMinimumSize(sz)
        self.update()
    def paintEvent(self, event):
        qp = QtGui.QPainter()
        qp.begin(self)
        if self.image:
            qp.drawImage(QtCore.QPoint(0, 0), self.image)
        qp.end()
class MyWindowClass(QMainWindow, form_class):
    def __init__(self, parent=None):
        QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.startButton.clicked.connect(self.start_clicked)
        self.window_width = self.ImgWidget.frameSize().width()
        self.window_height = self.ImgWidget.frameSize().height()
        self.ImgWidget = OwnImageWidget(self.ImgWidget)
        self.timer = QtCore.QTimer(self)
        self.timer.timeout.connect(self.update_frame)
        self.timer.start(1)
    def start_clicked(self):
        global running
        running = True
        capture_thread.start()
        self.startButton.setEnabled(False)
        self.startButton.setText('Starting...')
    def update_frame(self):
        if not q.empty():
            self.startButton.setText('Camera is live')
            frame = q.get()
            img = frame["img"]
            img_height, img_width, img_colors = img.shape
```

```
                scale_w = float(self.window_width) / float(img_width)
                scale_h = float(self.window_height) / float(img_height)
                scale = min([scale_w, scale_h])
                if scale == 0:
                    scale = 1
                img = cv2.resize(img, None, fx=scale, fy=scale, interpolation=cv2.INTER_CUBIC)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                height, width, bpc = img.shape
                bpl = bpc * width
                image = QtGui.QImage(img.data, width, height, bpl, QtGui.QImage.Format_RGB888)
                self.ImgWidget.setImage(image)
    def closeEvent(self, event):
        global running
        running = False
capture_thread = threading.Thread(target=grab, args=(0, q, 1920, 1080, 30))
app = QApplication(sys.argv)
w = MyWindowClass(None)
w.setWindowTitle('Kurokesu PyQT OpenCV USB camera test panel')
w.show()
app.exec_()
```

Simple.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>639</width>
    <height>504</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <property name="styleSheet">
   <string notr="true"/>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QGroupBox" name="groupBox">
    <property name="geometry">
     <rect>
      <x>10</x>
      <y>80</y>
      <width>621</width>
      <height>381</height>
     </rect>
    </property>
    <property name="sizePolicy">
     <sizepolicy hsizetype="Preferred" vsizetype="Preferred">
      <horstretch>0</horstretch>
      <verstretch>0</verstretch>
     </sizepolicy>
    </property>
    <property name="title">
     <string>Video</string>
    </property>
    <widget class="QWidget" name="ImgWidget" native="true">
     <property name="geometry">
      <rect>
       <x>10</x>
       <y>20</y>
       <width>601</width>
       <height>351</height>
      </rect>
     </property>
```

```
  </widget>
 </widget>
 <widget class="QPushButton" name="startButton">
  <property name="geometry">
   <rect>
    <x>10</x>
    <y>10</y>
    <width>621</width>
    <height>61</height>
   </rect>
  </property>
  <property name="text">
   <string>Start video</string>
  </property>
 </widget>
</widget>
<widget class="QMenuBar" name="menubar">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>639</width>
   <height>21</height>
  </rect>
 </property>
</widget>
<widget class="QStatusBar" name="statusbar"/>
</widget>
<resources/>
<connections/>
</ui>
```

[참고 2] Python-based sample code that interact with the Ryze Tello drone.

https://github.com/dji-sdk/Tello-Python

This toolkit contains three sample programs based on tello sdk and python2.7,including Single_Tello_Test, Tello_Video, and Tello_Video (With_Pose_Recognition). There is also a program file named tello_state.py.

- **Single_Tello_Test** : In Single_Tello_Test,You can design a series of command combinations by writing a txt script to let tello execute a series of actions you have designed. This program can also be used as a command set test tool for tello.
- **Tello_Video** : In Tello_Video， You can receive the video stream data from tello, decode the video through the h264 decoding library, and display it on a GUI interface based on Tkinter and PIL. In addition, it also supports a control panel that can operate tello. This sample code provides an example of receiving and processing and getting the correct video data. The source code of the h264 decoding library is also provided in the package, which can be used for your reference.
- **Tello_Video(With_Pose_Recognition)** : Tello_Video_With_Pose_Recognition is an application version modified from Tello_Video.It uses the decoded video data， and everytime extract a single frame image for pose recognition operation , and binds the specific posture and aircraft control commands to realize the pose control of Tello.This code is mainly used as an application case for utilizing the decoded video data of tello for image processing.
- **Tello_state.py** : Tello_state.py can read the various status data of tello, and can be used as a tool to debug and view the status of tello.

1. Tello_Video

This is an example using the Tello SDK v1.3.0.0 and above to receive video stream from Tello camera,decode the video stream and show the image by GUI.

Written in Python 2.7
Tello SDK v1.3.0.0 and above(with h.264 video streaming)
This example includes a simple UI build with Tkinter to interact with Tello
Interactive control of Tello based on human movement is achieved via body pose recognition module.

◯ Prerequisites
- ✓ Python2.7
- ✓ pip
- ✓ Python OpenCV
- ✓ Numpy
- ✓ PIL
- ✓ libboost-python
- ✓ Tkinter
- ✓ Python h264 decoder

◯ Install – Linux 일괄 설치 : 버전과 이전 설치 관계로 수정 설치
chmod +x linux_install.sh
./linux_install.sh

```
sudo apt-get update -y

sudo pip install --upgrade pip

# install cmake
#sudo apt-get install cmake -y
sudo pip install cmake

# install dependencies
```

```
sudo apt-get install libboost-all-dev -y
sudo apt-get install libavcodec-dev -y
sudo apt-get install libswscale-dev -y
sudo apt-get install python-numpy -y
sudo apt-get install python-matplotlib -y
sudo pip install opencv-python
sudo apt-get install python-imaging-tk

# pull and build h264 decoder library
cd h264decoder
mkdir build
cd build
cmake ..
make

# copy source .so file to tello.py directory
cp libh264decoder.so ../../
```