

# CH. 1. GUI 프로그래밍

# Section 01 PyQt 소개

---

## □ Qt 개요

- Qt는 본래 C++ 언어로 만들어진 GUI 킷 라이브러리이며 MS 윈도우, 리눅스, 맥 OS 등 다양한 플랫폼에서 동작하도록 만들어졌다.
- 특히, 리눅스의 대표적 데스크톱 환경 중 하나인 KDE의 GUI 기반 라이브러리로 사용되고 있다.
- Qt는 처음에 노르웨이 트롤테크(Trolltech)에서 개발되었으나, 현재는 노키아(Nokia)를 거쳐 디지아(Digia)에서 관리하고 있다.

# Section 01 PyQt 소개

---

## □ Qt 개요

- Qt의 주요 특징은 다음과 같다.
  - Qt는 C++를 기반으로 구현되어 객체지향 프로그래밍이 가능하며, 시그널과 슬롯이라는 이벤트 처리 방식을 제공한다.
  - 또한, 풍부한 GUI 위젯과 유니코드(Unicode) 및 il8n 등의 다양한 언어를 지원하며, SQL 데이터베이스, XML 파싱, 네트워크 연결, 멀티미디어 처리, 스레드 관리 등 여러 기능을 포함한다.
  - Qt GUI 프로그램을 개발하는 데 필요한 프로그래밍 도구로는 Qt Designer GUI 생성 프로그램과 Qt Creator 통합개발환경이 대표적이다.
  - Qt는 처음에 C++로 만들어졌지만, PySide6나 PyQt5와 같은 파이썬 바인딩 라이브러리가 있고, Qt Quick/QML이라는 절차 언어 개발 방식도 지원한다.

# Section 01 PyQt 소개

## □ Qt 개요

- Qt를 구성하는 세부적인 요소들은 표와 같다.
- 이 중에서도 특히 QtCore, QtWidgets 라이브러리에 대해 좀 더 자세히 살펴보자.

라이브러리	설명
QtCore	GUI를 제외한 기본 클래스들
QtGui	윈도우 이벤트, 2D GL, 이미지, 폰트, 텍스트 클래스 포함
QtWidgets	대부분의 GUI 클래스 포함
QtNetwork	UDP, TCP, FTP, HTTP 등 네트워크 관련 클래스들
QtSql	오픈소스 SQL DB와 통합된 클래스들
QtXml	XML 파서를 통해 SAX와 DOM 인터페이스 구현
QtMultimedia	멀티미디어 지원
QtUiTools	실행 시간 GUI 생성 지원
QtQuick	QML 기반의 절차 언어 지원
QtWebkit	웹 브라우저 엔진 지원
QtTest	Unit 테스트 지원

# Section 01 PyQt 소개

## □ PyQt 설치

### • 새 프로젝트 생성

PC 새 프로젝트

순수 Python  
Python  
Django  
> 기타

이름: Basic\_GUI

위치: D:\Projects

프로젝트가 다음의 위치에 생성됩니다. D:\Projects\Basic\_GUI

☐ Git 저장소 생성 ☐ 환영 인사 스크립트 생성

인터프리터 타입: 프로젝트 venv 기본 Conda 사용자 지정 환경

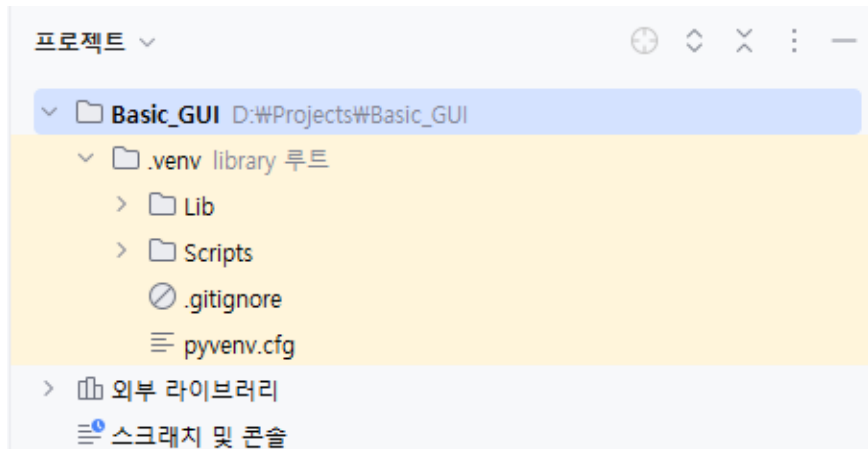
Python 버전: C:/Users/jinu/AppData/Local/Programs/Python/Python312/python.exe 시스템에서

Python 가상 환경이 프로젝트 루트에 생성됩니다. D:\Projects\Basic\_GUI\venv

# Section 01 PyQt 소개

## □PyQt 설치

- 프로젝트 탐색창 열기 (Alt + 1)



전체 검색 Shift 두 번

파일로 이동 Ctrl+Shift+N

최근 파일 Ctrl+E

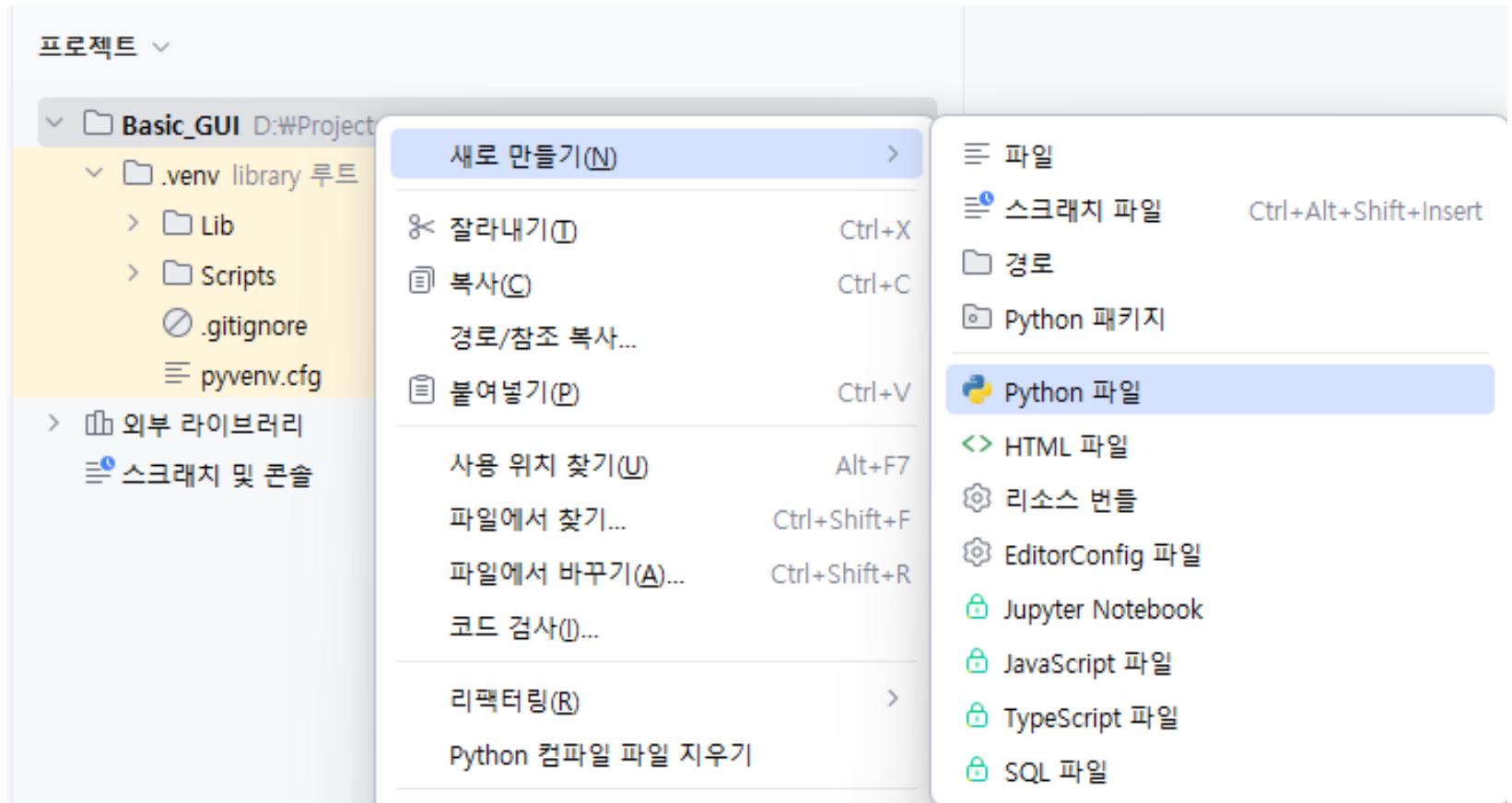
내비게이션 바 Alt+Home

파일을 여기에 드롭하여 열기

# Section 01 PyQt 소개

## □PyQt 설치

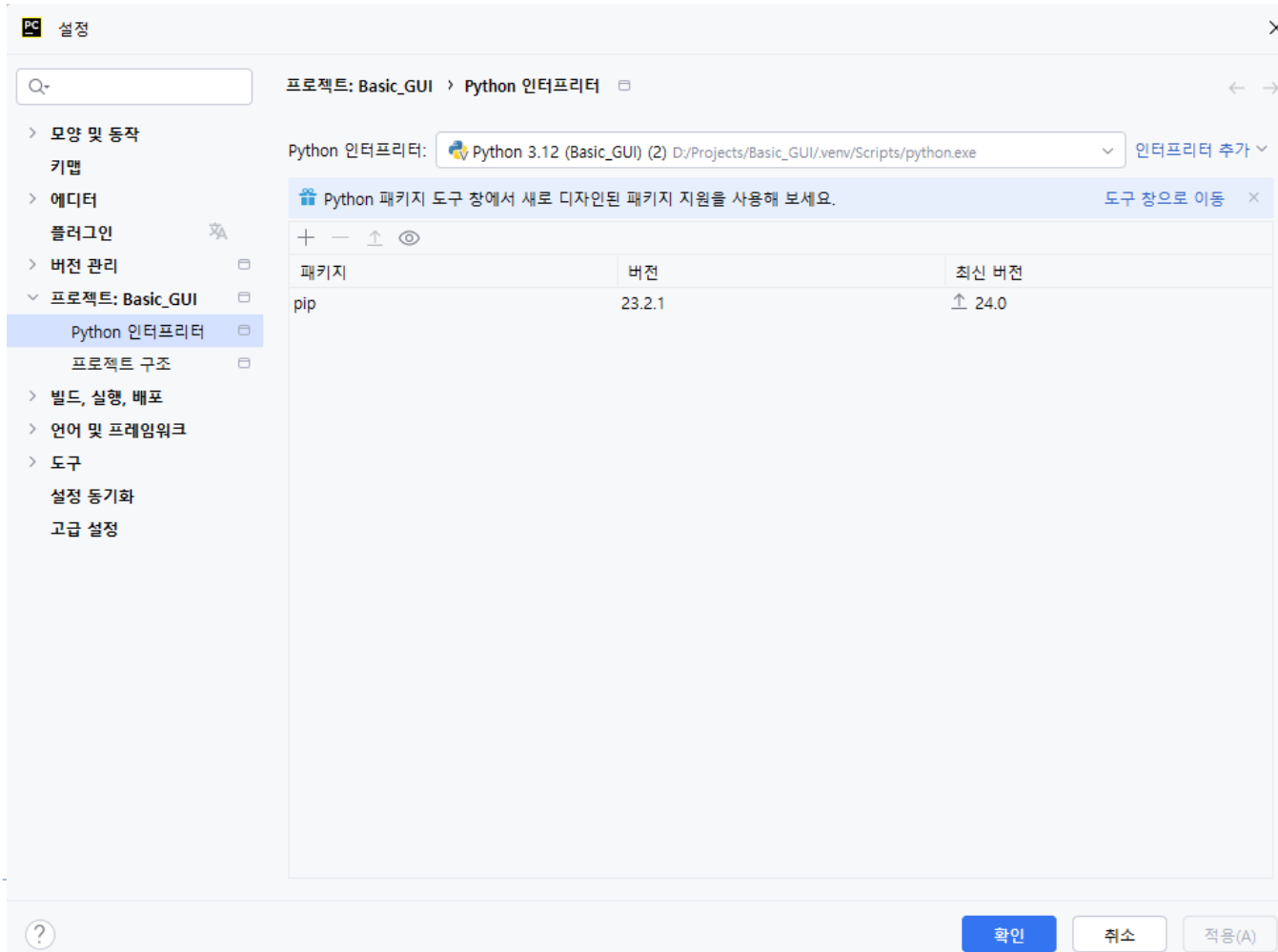
### • 새 파일 생성



# Section 01 PyQt 소개

## □PyQt 설치

- 파이참에서 파이썬 외부모듈 설치 (Ctrl + Alt + s)

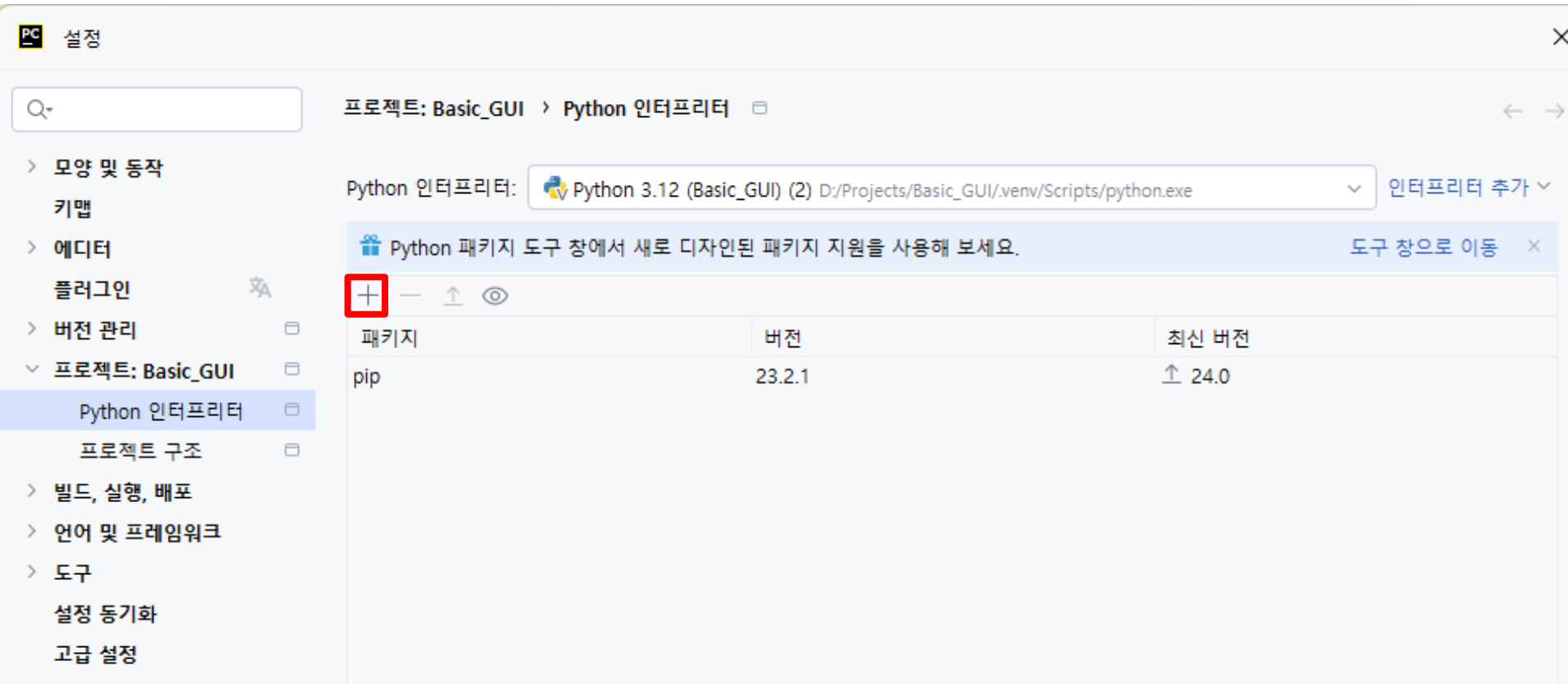




# Section 01 PyQt 소개

## □PyQt 설치

- 파이참에서 파이썬 외부모듈 설치 (Ctrl + Alt + s)



PC 설정

프로젝트: Basic\_GUI > Python 인터프리터

Python 인터프리터: Python 3.12 (Basic\_GUI) (2) D:/Projects/Basic\_GUI/.venv/Scripts/python.exe 인터프리터 추가

Python 패키지 도구 창에서 새로 디자인된 패키지 지원을 사용해 보세요. 도구 창으로 이동

패키지	버전	최신 버전
pip	23.2.1	↗ 24.0



설명

PyQt-server-client

PyQt3D

PyQt3D-Qt

PyQt3D-Qt5

PyQt4

PyQt4Enhanced

PyQt4\_windows\_whl

PyQt5

PyQt5-Auto

PyQt5-Frameless-Window

PyQt5-Qt

PyQt5-Qt5

PyQt5-comet

PyQt5-python-mess-client

PyQt5-python-mess-server

PyQt5-sip

PyQt5-stubs

PyQt5Designer

PyQt5Singleton

PyQt6

PyQt6-3D

Python bindings for the Qt cross platform application toolkit

<mailto:Riverbank Computing Limited>☐ 버전 지정

6.7.0

☐ 옵션

패키지 설치(I)

닫기(C)

## Section 02 기초

---

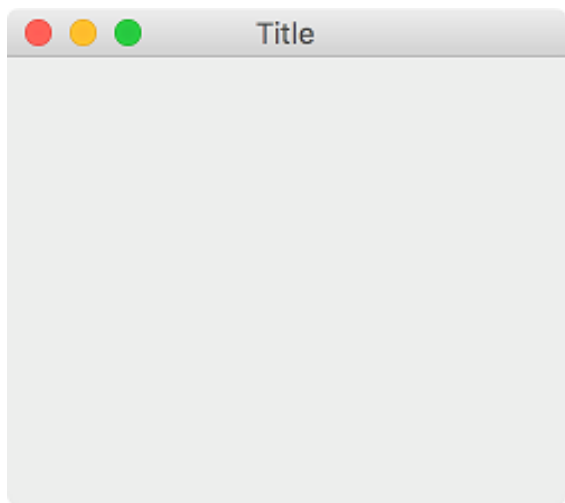
- 먼저 간단한 프로그램을 만들어보자.
- 몇 줄의 코드를 작성해보면서 아래에 나와있는 여러가지 GUI 프로그램의 기초를 이해할 수 있다.
- 순서는 아래와 같다.
  - 창 띄우기
  - 어플리케이션 아이콘 넣기
  - 창 닫기
  - 툴팁 나타내기
  - 상태바 만들기
  - 메뉴바 만들기
  - 툴바 만들기
  - 창을 화면의 가운데 띄우기
  - 날짜와 시간 표시하기

# Section 02 기초

---

## □창 띄우기

- 그림과 같은 작은 창을 하나 띄워보자.
- 창의 오른쪽 위 (Windows) 또는 왼쪽 위 (macOS)에 기본적으로 제공되는 버튼들로 창의 크기를 최대화, 최소화하거나 종료할 수 있다.
- 또한 마우스를 가지고 창을 이동하거나 창의 크기를 조절할 수 있다.
- 이러한 기능은 사실 많은 코드를 필요로 하지만 대부분의 어플리케이션에서 자주 사용되는 기능이기에 이미 누군가가 코드로 만들어 놓았다.



# Section 02 기초

## □창 띄우기

- ex\_1\_1.py

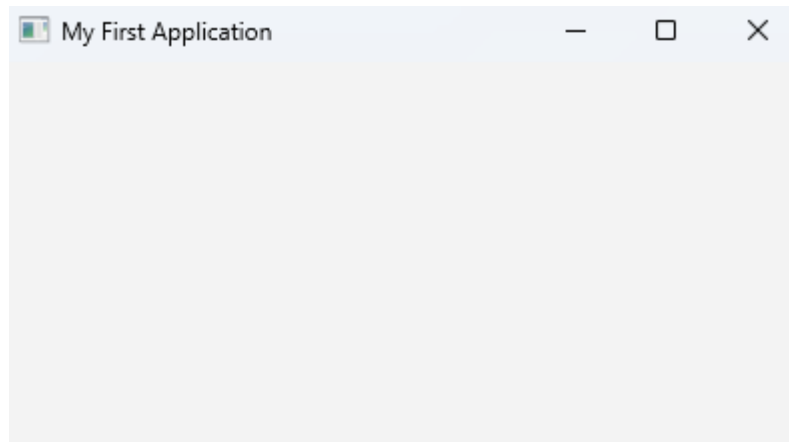
```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget
3
4  class MyApp(QWidget):
5      def __init__(self):
6          super().__init__()
7          self.initUI()
8
9      def initUI(self):
10         self.setWindowTitle('My First Application')
11         self.move(300, 300)
12         self.resize(400, 200)
13         self.show()
14
```

# Section 02 기초

## □창 띄우기

- ex\_1\_1.py

```
15 if __name__ == '__main__':  
16     app = QApplication(sys.argv)  
17     ex = MyApp()  
18     sys.exit(app.exec())
```



# Section 02 기초

---

## □창 띄우기

```
import sys  
from PyQt6.QtWidgets import QApplication, QWidget
```

- **필요한 모듈들을 불러온다.**
- **기본적인 UI 구성요소를 제공하는 위젯 (클래스)들은 PyQt6.QtWidgets 모듈에 포함되어 있다.**
- **QtWidgets 모듈에 포함된 모든 클래스들과 이에 대한 자세한 설명은 QtWidgets 공식 문서에서 확인할 수 있다.**

## Section 02 기초

---

### □창 띄우기

```
self.setWindowTitle('My First Application')  
self.move(300, 300)  
self.resize(400, 200)  
self.show()
```

- 여기서 self는 MyApp 객체를 말한다.
- setTitle() 메서드는 타이틀바에 나타나는 창의 제목을 설정한다.
- move() 메서드는 위젯을 스크린의 x=300px, y=300px의 위치로 이동시킨다.
- resize() 메서드는 위젯의 크기를 너비 400px, 높이 200px로 조절한다.
- show() 메서드는 위젯을 스크린에 보여준다.



## Section 02 기초

### □창 띄우기

```
if __name__ == '__main__':
```

- **'\_\_name\_\_'은 현재 모듈의 이름이 저장되는 내장 변수이다.**
- 만약 ' moduleA.py ' 라는 코드를 import해서 예제 코드를 수행하면 **\_\_name\_\_ 은 ' moduleA ' 가 된다.**
- **그렇지 않고 코드를 직접 실행한다면 \_\_name\_\_ 은 \_\_main\_\_ 이 된다.**
- 따라서 이 한 줄의 코드를 통해 프로그램이 직접 실행되는지 혹은 모듈을 통해 실행되는지를 확인한다.

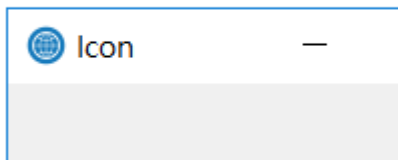
```
app = QApplication(sys.argv)
```

- 모든 PyQt5 어플리케이션은 어플리케이션 객체를 생성해야 한다.
- QApplication 클래스에 대한 자세한 설명은 QApplication 공식 문서에서 확인할 수 있다.

## Section 02 기초

### □ 어플리케이션 아이콘 넣기

- 어플리케이션 아이콘은 타이틀바의 왼쪽 끝에 보여질 작은 이미지이다.
- 어플리케이션 아이콘을 표시하는 방법을 소개한다.
- 우선 폴더 안에, 아래와 같이 아이콘으로 사용할 이미지 파일을 저장해 둔다.



- 다음 예제는 창을 하나 띄우면서, 타이틀바의 왼쪽에 작은 아이콘이 나타나도록 했다.

## Section 02 기초

### □ 어플리케이션 아이콘 넣기

- ex\_1\_2.py

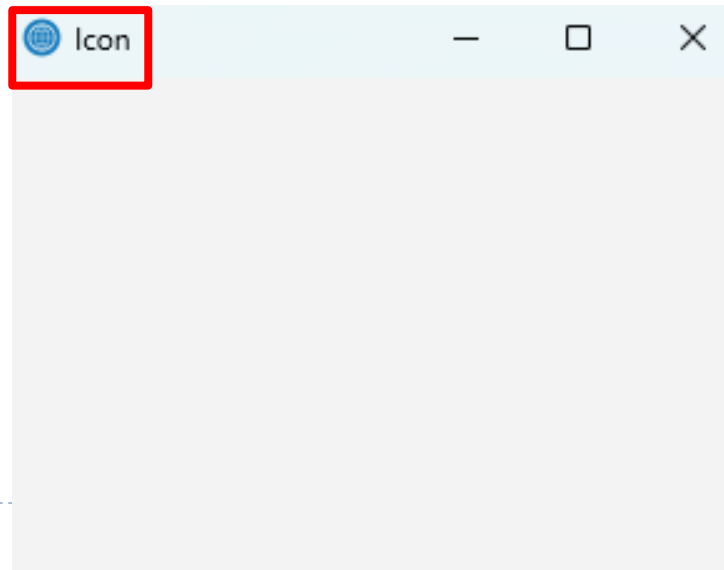
```
1 import sys
2 from PyQt6.QtWidgets import QApplication, QWidget
3 from PyQt6.QtGui import QIcon
4
5 class MyApp(QWidget):
6     def __init__(self):
7         super().__init__()
8         self.initUI()
9
10    def initUI(self):
11        self.setWindowTitle('Icon')
12        self.setWindowIcon(QIcon('./images/web.png'))
13        self.setGeometry(300, 300, 300, 200)
14        self.show()
```

## Section 02 기초

### □ 어플리케이션 아이콘 넣기

- ex\_1\_2.py

```
15  
16     if __name__ == '__main__':  
17         app = QApplication(sys.argv)  
18         ex = MyApp()  
19         app.exec()
```



## Section 02 기초

### □ 어플리케이션 아이콘 넣기

```
self.setWindowIcon(QIcon('./images/web.png'))
```

- setWindowIcon() 메서드는 어플리케이션 아이콘을 설정하도록 한다.
- 이를 위해서 QIcon 객체를 생성하였다.
- QIcon()에 보여질 이미지('./images/web.png')를 입력한다.
- 이미지 파일을 다른 폴더에 따로 저장해 둔 경우에는 경로까지 함께 입력하면 된다.

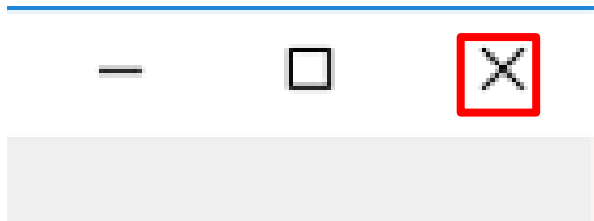
```
self.setGeometry(300, 300, 300, 200)
```

- setGeometry() 메서드는 창의 위치와 크기를 설정한다.
- 앞의 두 매개변수는 창의 x, y 위치를 결정하고, 뒤의 두 매개변수는 각각 창의 너비와 높이를 결정한다.
- 이 메서드는 창 띄우기 예제에서 사용했던 move()와 resize() 메서드를 하나로 합쳐놓은 것과 같다.

## Section 02 기초

### □창 닫기

- 창을 닫는 가장 간단한 방법은 타이틀바의 오른쪽 (Windows) 또는 왼쪽 (macOS) 'X' 버튼을 클릭하는 것이다.
- 이번에는 프로그래밍을 통해 창을 닫는 법을 알아보자.
- 시그널 (Signal)과 슬롯 (Slot)에 대해서도 간단하게 다뤄볼 것이다.



# Section 02 기초

## □창 달기

- ex\_1\_3.py

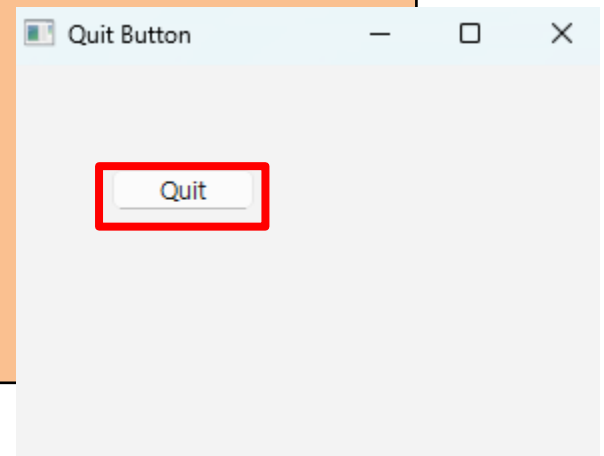
```
1 import sys
2 from PyQt6.QtWidgets import QApplication, QWidget, QPushButton
3 from PyQt6.QtCore import QApplication
4
5 class MyApp(QWidget):
6     def __init__(self):
7         super().__init__()
8         self.initUI()
9
10    def initUI(self):
11        btn = QPushButton('Quit', self)
12        btn.move(50, 50)
13        btn.resize(btn.sizeHint())
14        btn.clicked.connect(QCoreApplication.instance().quit)
```

## Section 02 기초

### □창 달기

- ex\_1\_3.py

```
15
16     self.setWindowTitle('Quit Button')
17     self.setGeometry(300, 300, 300, 200)
18     self.show()
19
20 if __name__ == '__main__':
21     app = QApplication(sys.argv)
22     ex = MyApp()
23     app.exec()
```



- 'Quit' 버튼을 만들었다.
- 이제 이 버튼을 클릭하면 어플리케이션이 종료된다.



## Section 02 기초

### □창 달기

```
from PyQt5.QtCore import QApplication
```

- QtCore **모듈의** QApplication **클래스를 불러온다.**

```
btn = QPushButton('Quit', self)
```

- **푸시버튼을 하나 만든다.**
- **이 버튼 (btn)은 QPushButton 클래스의 인스턴스이다.**
- **생성자 (QPushButton())의 첫 번째 파라미터에는 버튼에 표시될 텍스트를 입력하고, 두 번째 파라미터에는 버튼이 위치할 부모 위젯을 입력한다.**
- **푸시버튼 위젯에 대한 자세한 설명은 QPushButton 페이지를 참고할 것.**

## Section 02 기초

### □창 달기

```
btn.clicked.connect(QCoreApplication.instance().quit)
```

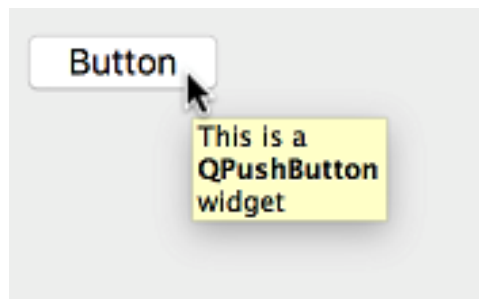
- PyQt6에서의 이벤트 처리는 시그널과 슬롯 메커니즘으로 이루어진다.
- 버튼 (btn)을 클릭하면 ' clicked ' 시그널이 만들어진다.
- instance() 메서드는 현재 인스턴스를 반환한다.
- 'clicked' 시그널은 어플리케이션을 종료하는 quit() 메서드에 연결된다.
- 이렇게 발신자 (Sender)와 수신자 (Receiver), 두 객체 간에 커뮤니케이션이 이루어진다.
- 이 예제에서 발신자는 푸시버튼 (btn)이고, 수신자는 어플리케이션 객체 (app)이다.

## Section 02 기초

---

### □툴팁 나타내기

- 툴팁은 어떤 위젯의 기능을 설명하는 등의 역할을 하는 말풍선 형태의 도움말이다. (QToolTip 공식 문서 참고)
- 위젯에 있는 모든 구성 요소에 대해서 툴팁(tooltip)이 나타나도록 할 수 있다.
- 이번에는 `setToolTip()` 메서드를 이용해서 위젯에 툴팁을 만들어 보자.



## Section 02 기초

### ☐ 툴팁 나타내기

- ex\_1\_4.py

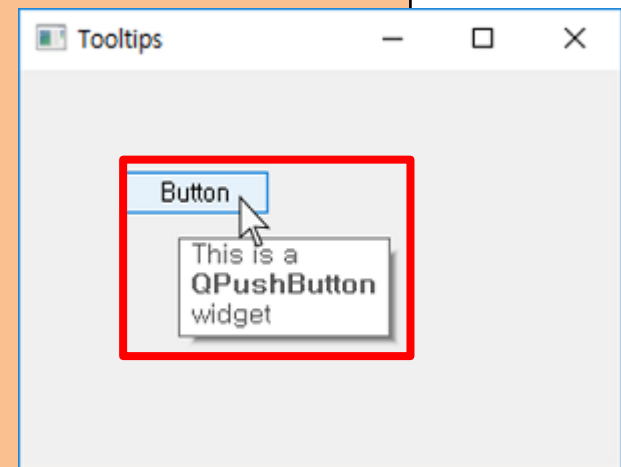
```
1 import sys
2 from PyQt6.QtWidgets import QApplication, QWidget, QPushButton, QToolTip
3 from PyQt6.QtGui import QFont
4
5 class MyApp(QWidget):
6     def __init__(self):
7         super().__init__()
8         self.initUI()
9
10    def initUI(self):
11        QToolTip.setFont(QFont('SansSerif', 10))
12        self.setToolTip('This is a <b>QWidget</b> widget')
13
14        btn = QPushButton('Button', self)
```

## Section 02 기초

### □ 툴팁 나타내기

- ex\_1\_4.py

```
15 btn.setToolTip("This is a <b>QPushButton</b> widget")
16 btn.move(50, 50)
17 btn.resize(btn.sizeHint())
18
19 self.setWindowTitle("Tooltips")
20 self.setGeometry(300, 300, 300, 200)
21 self.show()
22
23 if __name__ == '__main__':
24     app = QApplication(sys.argv)
25     ex = MyApp()
26     sys.exit(app.exec())
```



- ▶ 29 • 푸시버튼(btn)과 창(MyApp) 위젯에 마우스를 올리면 각각 설정한 텍스트가 툴팁으로 나타난다.

## Section 02 기초

### □ 툴팁 나타내기

```
QToolTip.setFont(QFont('SansSerif', 10))  
self.setToolTip('This is a <b>QWidget</b> widget')
```

- 먼저 툴팁에 사용될 폰트를 설정한다.
- 여기에서는 10px 크기의 ' SansSerif ' 폰트를 사용한다.
- 툴팁을 만들기 위해서는 `setToolTip()` 메서드를 사용해서, 표시될 텍스트를 입력해준다.

```
btn = QPushButton('Button', self)  
btn.setToolTip('This is a <b>QPushButton</b> widget')
```

- 푸시버튼을 하나 만들고, 이 버튼에도 툴팁을 달아줍니다.

## Section 02 기초

---

### □ 툴팁 나타내기

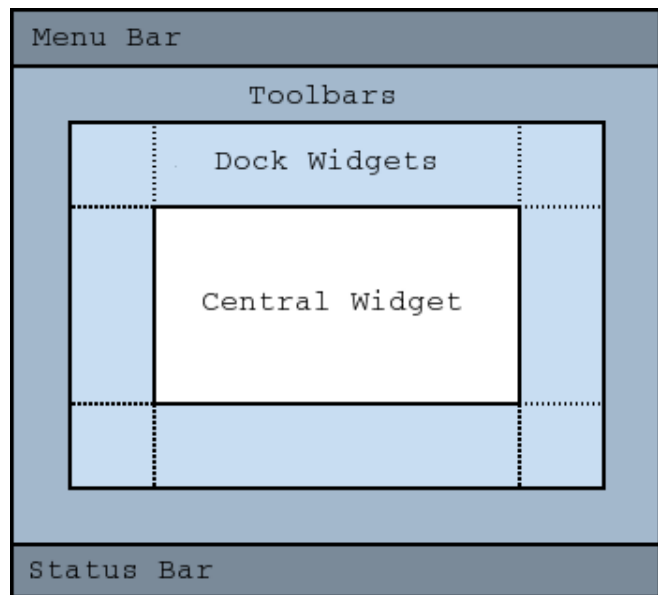
```
btn.move(50, 50)  
btn.resize(btn.sizeHint())
```

- **버튼의 위치와 크기를 설정한다.**
- **sizeHint() 메서드는 버튼을 적절한 크기로 설정하도록 도와준다.**

## Section 02 기초

### □ 상태바 만들기

- 메인창(Main window)은 메뉴바, 툴바, 상태바를 갖는 전형적인 어플리케이션 창이다. (QMainWindow 공식 문서 참고)



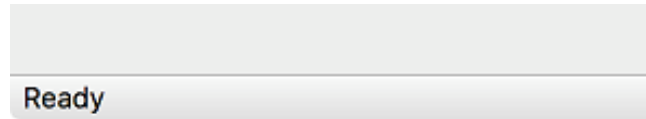
- 메인창은 QMenuBar, QToolBar, QDockWidget, QStatusBar를 위한 고유의 레이아웃을 갖고 있다.
- 또한 가운데 영역에 중심위젯 (Central widget)을 위한 영역을 갖고 있다.



## Section 02 기초

### □ 상태바 만들기

- QMainWindow 클래스를 이용해서 메인 어플리케이션 창을 만들 수 있다.



- 우선 QStatusBar를 이용해서 메인 창에 상태바 (status bar)를 하나 만들어 보자.
- 상태바는 어플리케이션의 상태를 알려주기 위해 어플리케이션의 하단에 위치하는 위젯이다. (QStatusBar 공식 문서 참고)
- 상태바에 텍스트를 표시하기 위해서는 showMessage() 메서드를 사용한다.
- 텍스트가 사라지게 하고 싶으면, clearMessage() 메서드를 사용하거나, showMessage() 메서드에 텍스트가 표시되는 시간을 설정할 수 있다.
- 현재 상태바에 표시되는 메시지 텍스트를 갖고 오고 싶을 때는 currentMessage() 메서드를 사용한다.
- QStatusBar 클래스는 상태바에 표시되는 메시지가 바뀔 때 마다 messageChanged() 시그널을 발생한다.

# Section 02 기초

## □ 상태바 만들기

- ex\_1\_5.py

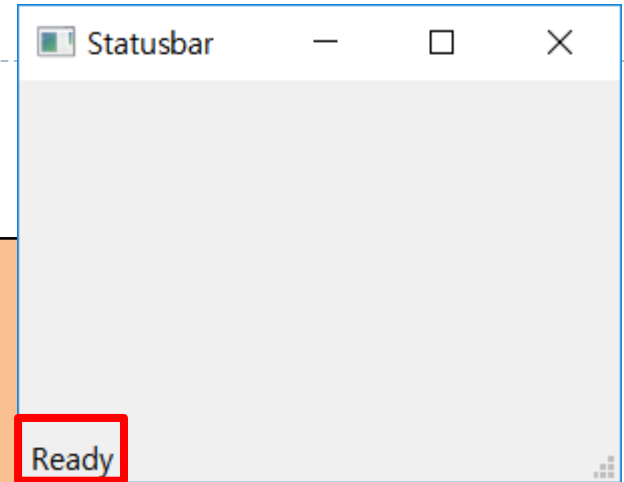
```
1 import sys
2 from PyQt6.QtWidgets import QApplication, QMainWindow
3
4 class MyApp(QWidget):
5     def __init__(self):
6         super().__init__()
7         self.initUI()
8
9     def initUI(self):
10         self.statusBar().showMessage('Ready')
11
12         self.setWindowTitle('Statusbar')
13         self.setGeometry(300, 300, 300, 200)
14         self.show()
```

## Section 02 기초

### □ 상태바 만들기

- ex\_1\_5.py

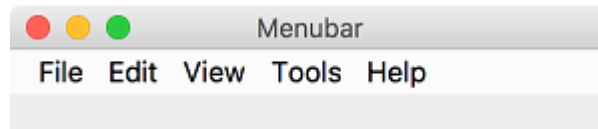
```
15  
16     if __name__ == '__main__':  
17         app = QApplication(sys.argv)  
18         ex = MyApp()  
19         sys.exit(app.exec())
```



- 이제 어플리케이션 창의 아래쪽에 상태바가 하나 표시된다.

## Section 02 기초

### □메뉴바 만들기



- GUI 어플리케이션에서 메뉴바(menu bar)는 흔하게 사용된다.
- 다양한 명령들의 모음이 메뉴바에 위치한다. (QMenuBar 공식 문서 참고)
- macOS에서는 메뉴바를 다르게 다루는데, 아래 예제에서 볼 수 있듯이 한 줄의 코드(`menubar.setNativeMenuBar(False)`)를 추가함으로써 macOS에서도 Windows 환경과 동일한 결과를 얻을 수 있다.
- 우선 폴더 안에 아래와 같이 메뉴에 해당하는 아이콘(exit.png)을 저장해 둔다.



# Section 02 기초

## □메뉴바 만들기

- ex\_1\_6.py

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QMainWindow
3  from PyQt6.QtGui import QIcon, QAction
4
5  class MyApp(QMainWindow):
6      def __init__(self):
7          super().__init__()
8          self.initUI()
9
10     def initUI(self):
11         exitAction = QAction(QIcon('./images/exit.png'), 'Exit', self)
12         exitAction.setShortcut('Ctrl+Q')
13         exitAction.setStatusTip('Exit application')
14         exitAction.triggered.connect(QApplication.instance().quit)
```

## Section 02 기초

### □메뉴바 만들기

- ex\_1\_6.py

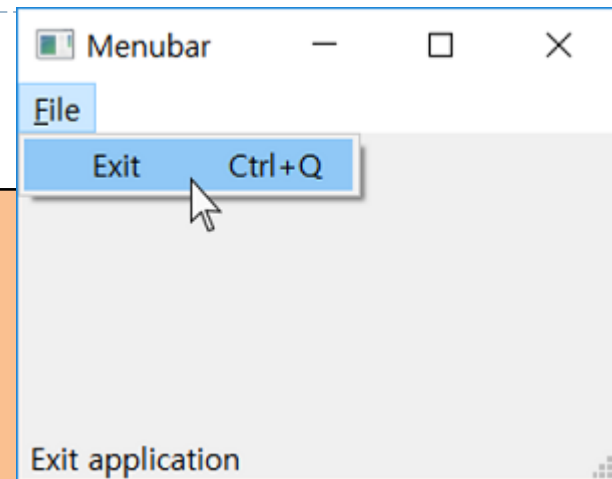
```
15
16     self.statusBar()
17
18     menubar = self.menuBar()
19     menubar.setNativeMenuBar(False)
20     filemenu = menubar.addMenu('&File')
21     filemenu.addAction(exitAction)
22
23     self.setWindowTitle('Menubar')
24     self.setGeometry(300, 300, 300, 200)
25     self.show()
26
```

## Section 02 기초

### □ 메뉴바 만들기

- ex\_1\_6.py

```
27 if __name__ == '__main__':  
28     app = QApplication(sys.argv)  
29     ex = MyApp()  
30     sys.exit(app.exec())
```



- 한 개의 메뉴를 갖는 메뉴바를 만들었다.
- 이 메뉴는 클릭했을 때 어플리케이션을 종료하는 기능을 갖고 있다.
- 또한 이 기능은 단축키 (Ctrl+Q)로도 실행이 가능하다.

## Section 02 기초

### □ 메뉴바 만들기

```
exitAction = QAction(QIcon('./images/exit.png'), 'Exit', self)
exitAction.setShortcut('Ctrl+Q')
exitAction.setStatusTip('Exit application')
```

- 이 세 줄의 코드를 통해 아이콘 (exit.png)과 'Exit' 라벨을 갖는 하나의 동작 (action)을 만들고, 이 동작에 대해 단축키 (shortcut)를 정의한다.
- 또한 메뉴에 마우스를 올렸을 때, 상태바에 나타날 상태팁을 setStatusTip() 메서드를 사용하여 설정했다.

```
exitAction.triggered.connect(QApplication.quit)
```

- 이 동작을 선택했을 때, 생성된 (triggered) 시그널이 QApplication 위젯의 quit() 메서드에 연결되고, 어플리케이션을 종료시키게 된다.



## Section 02 기초

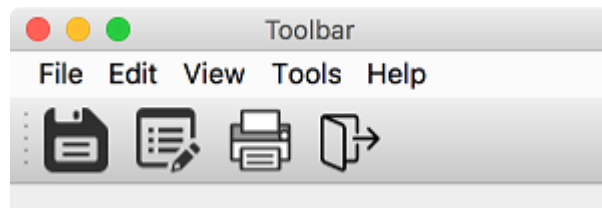
### □ 메뉴바 만들기

```
menubar = self.menuBar()  
menubar.setNativeMenuBar(False)  
fileMenu = menubar.addMenu('&File')  
fileMenu.addAction(exitAction)
```

- `menuBar()` 메서드는 메뉴바를 생성한다.
- 이어서 'File' 메뉴를 하나 만들고, 거기에 'exitAction' 동작을 추가한다.
- ' &File ' 의 앰퍼샌드 (ampersand, &)는 간편하게 단축키를 설정하도록 해준다.
- ' F ' 앞에 앰퍼샌드가 있으므로 ' Alt+F ' 가 File 메뉴의 단축키가 된다.
- 만약 ' l ' 의 앞에 앰퍼샌드를 넣으면 ' Alt+l ' 가 단축키가 된다.

## Section 02 기초

### □툴바 만들기



- 메뉴(menu)가 어플리케이션에서 사용되는 모든 명령의 모음이라면, 툴바(toolbar)는 자주 사용하는 명령들을 더 편리하게 사용할 수 있도록 해준다. (QToolBar 공식 문서 참고)
- 폴더 안에 툴바의 각 기능에 해당하는 아이콘들을 저장해 둔다.



save.png, edit.png, print.png, exit.png

# Section 02 기초

## □ 툴바 만들기

- ex\_1\_7.py

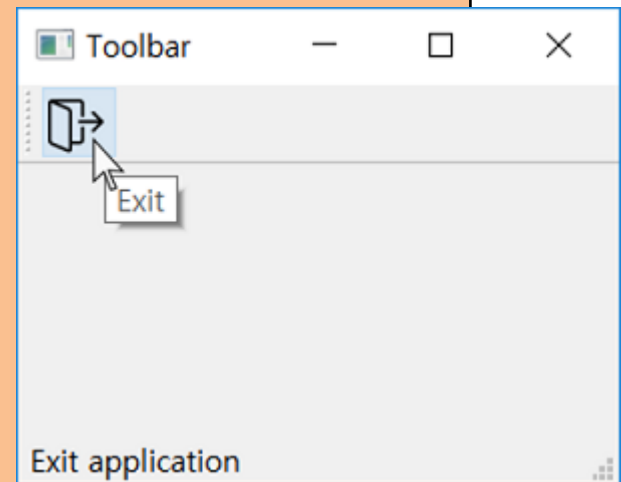
```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QMainWindow
3  from PyQt6.QtGui import QIcon, QAction
4
5  class MyApp(QMainWindow):
6      def __init__(self):
7          super().__init__()
8          self.initUI()
9
10     def initUI(self):
11         exitAction = QAction(QIcon('./images/exit.png'), 'Exit', self)
12         exitAction.setShortcut('Ctrl+Q')
13         exitAction.setStatusTip('Exit application')
14         exitAction.triggered.connect(QApplication.instance().quit)
```

# Section 02 기초

## □ 툴바 만들기

- ex\_1\_7.py

```
15
16     self.statusBar()
17
18     self.toolbar = self.addToolBar('Exit')
19     self.toolbar.addAction(exitAction)
20
21     self.setWindowTitle('Toolbar')
22     self.setGeometry(300, 300, 300, 200)
23     self.show()
24
25
26 if __name__ == '__main__':
27     app = QApplication(sys.argv)
28     ex = MyApp()
29     app.exec()
```



## Section 02 기초

### □툴바 만들기

```
exitAction = QAction(QIcon('exit.png'), 'Exit', self)
exitAction.setShortcut('Ctrl+Q')
exitAction.setStatusTip('Exit application')
exitAction.triggered.connect(qApp.quit)
```

- 메뉴바의 경우와 마찬가지로 QAction 객체를 하나 생성한다.
- 이 객체는 아이콘 (exit.png), 라벨 ('Exit')을 포함하고, 단축키 (Ctrl+Q)를 통해 실행 가능하다.
- 상태바에 메시지 ( ' Exit application ' )를 보여주고, 클릭 시 생성되는 시그널은 quit() 메서드에 연결되어 있다.

## Section 02 기초

---

### □툴바 만들기

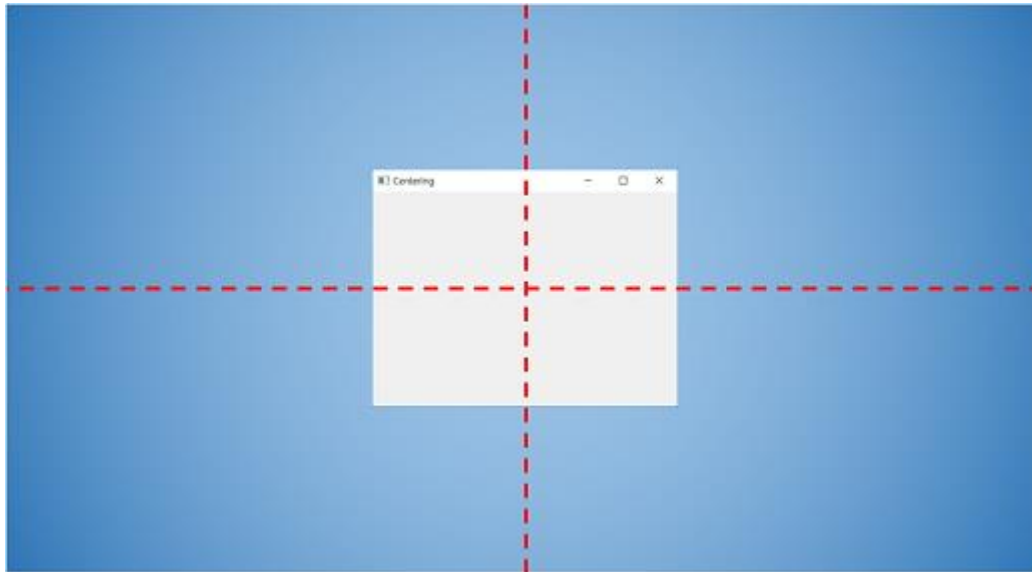
```
self.toolbar = self.addToolBar('Exit')  
self.toolbar.addAction(exitAction)
```

- **addToolBar()를 이용해서 투바를 만들고, addAction()을 이용해서 투바에 exitAction 동작을 추가했다.**

## Section 02 기초

---

### □창을 화면의 가운데로



- 위의 그림과 같이 창을 모니터 화면의 가운데에 띄워보자.

## Section 02 기초

### □창을 화면의 가운데로

- ex\_1\_8.py

```
1 import sys
2 from PyQt6.QtWidgets import QApplication, QMainWindow
3 from PyQt6.QtGui import QGuiApplication
4
5 class MyApp(QMainWindow):
6     def __init__(self):
7         super().__init__()
8         self.initUI()
9
10    def initUI(self):
11        self.setWindowTitle('Centering')
12        self.resize(500, 350)
13        self.center()
14        self.show()
```



## Section 02 기초

### □창을 화면의 가운데로

- ex\_1\_8.py

```
15
16     def center(self):
17         qr = self.frameGeometry()
18         cp = QtGuiApplication.primaryScreen().availableGeometry().center()
19         qr.moveCenter(cp)
20         self.move(qr.topLeft())
21
22     if __name__ == '__main__':
23         app = QApplication(sys.argv)
24         ex = MyApp()
25         app.exec()
```

- 창이 화면의 정가운데에 띄워진다.

## Section 02 기초

---

### □창을 화면의 가운데로

```
self.center()
```

- center() 메서드를 통해서 창이 화면의 가운데에 위치하게 된다.

```
self.center()
```

- frameGeometry() 메서드를 이용해서 창의 위치와 크기 정보를 가져온다.

```
cp = QtGuiApplication.primaryScreen().availableGeometry().center()
```

- 사용하는 모니터 화면의 가운데 위치를 파악한다.

## Section 02 기초

---

### □창을 화면의 가운데로

```
qr.moveCenter(cp)
```

- 창을 직사각형 위치를 화면의 중심의 위치로 이동한다.

```
self.move(qr.topLeft())
```

- 현재 창을, 화면의 중심으로 이동했던 직사각형(qr)의 위치로 이동시킨다.
- 결과적으로 현재 창의 중심이 화면의 중심과 일치하게 돼서 창이 가운데에 나타나게 된다.

## Section 02 기초

### □ 날짜와 시간 표시하기

- QtCore 모듈의 QDateTime, QDateTime, QDateTime 클래스를 이용해서 어플리케이션에 날짜와 시간을 표시할 수 있다.
- 날짜 표시하기(QDate)
  - QDate 클래스는 날짜와 관련된 기능들을 제공한다.
  - 우선 QDate 클래스를 이용해서 날짜를 출력해 보자.

```
>>> from PyQt6.QtCore import QDate
>>> Now = QDate.currentDate()
>>> print(now.toString())
```

- currentDate() 메서드는 현재 날짜를 반환한다.
- toString() 메서드를 통해 현재 날짜를 문자열로 출력할 수 있다.
- 결과는 아래와 같다.

```
Python Console>>> from PyQt6.QtCore import QDate
>>> now = QDate.currentDate()
>>> print(now.toString())
Sun May 5 2024
```

## Section 02 기초

### □ 날짜와 시간 표시하기

- 날짜 형식 설정하기

➤ `toString()` 메서드의 `format` 파라미터를 설정함으로써 날짜의 형식을 정할 수 있다.

```
>>> from PyQt6.QtCore import QDate, Qt
>>> now = QDate.currentDate()
>>> print(now.toString('d.M.yy'))
>>> print(now.toString('dd.MM.yyyy'))
>>> print(now.toString('ddd.MMMM.yyyy'))
```

➤ 'd'는 일(day), 'M'은 달(month), 'y'는 연도(year)를 나타낸다.

➤ 각 문자의 개수에 따라 날짜의 형식이 다르게 출력된다.

➤ `Qt.ISODate`, `Qt.DefaultLocaleLongDate`를 입력함으로써 ISO 표준 형식 또는 어플리케이션의 기본 설정에 맞게 출력할 수 있다.

# Section 02 기초

## □ 날짜와 시간 표시하기

### • 날짜 형식 설정하기

➤ 자세한 내용은 아래의 표 또는 공식 문서를 참고하면 된다.

Expression	Output
d	the day as number without a leading zero (1 to 31)
dd	the day as number with a leading zero (01 to 31)
ddd	the abbreviated localized day name (e.g. 'Mon' to 'Sun'). Uses the system locale to localize the name, i.e. <code>QLocale::system()</code> .
dddd	the long localized day name (e.g. 'Monday' to 'Sunday'). Uses the system locale to localize the name, i.e. <code>QLocale::system()</code> .
M	the month as number without a leading zero (1 to 12)
MM	the month as number with a leading zero (01 to 12)
MMM	the abbreviated localized month name (e.g. 'Jan' to 'Dec'). Uses the system locale to localize the name, i.e. <code>QLocale::system()</code> .
MMMM	the long localized month name (e.g. 'January' to 'December'). Uses the system locale to localize the name, i.e. <code>QLocale::system()</code> .
yy	the year as two digit number (00 to 99)
yyyy	the year as four digit number. If the year is negative, a minus sign is prepended in addition.

Constant	Value	Description
<code>Qt::TextDate</code>	0	The default Qt format, which includes the day and month name, the day number in the month, and the year in full. The day and month names will be short, localized names. This is basically equivalent to using the date format string, "ddd MMM d yyyy". See <a href="#">QDate::toString()</a> for more information.
<code>Qt::ISODate</code>	1	<a href="#">ISO 8601</a> extended format: either yyyy-MM-dd for dates or yyyy-MM-ddTHH:mm:ss (e.g. 2017-07-24T15:46:29), or with a time-zone suffix (Z for UTC otherwise an offset as [+ -]HH:mm) where appropriate for combined dates and times.
<code>Qt::ISODateWithMs</code>	9	<a href="#">ISO 8601</a> extended format, including milliseconds if applicable.
<code>Qt::SystemLocaleShortDate</code>	4	The <a href="#">short format</a> used by the <a href="#">operating system</a> .
<code>Qt::SystemLocaleLongDate</code>	5	The <a href="#">long format</a> used by the <a href="#">operating system</a> .
<code>Qt::DefaultLocaleShortDate</code>	6	The <a href="#">short format</a> specified by the <a href="#">application's locale</a> .
<code>Qt::DefaultLocaleLongDate</code>	7	The <a href="#">long format</a> used by the <a href="#">application's locale</a> .
<code>Qt::SystemLocaleDate</code>	2	<i>This enum value is deprecated.</i> Use <code>Qt::SystemLocaleShortDate</code> instead (or <code>Qt::SystemLocaleLongDate</code> if you want long dates).
<code>Qt::LocaleDate</code>	3	<i>This enum value is deprecated.</i> Use <code>Qt::DefaultLocaleShortDate</code> instead (or <code>Qt::DefaultLocaleLongDate</code> if you want long dates).
<code>Qt::LocalDate</code>	<code>SystemLocaleDate</code>	<i>This enum value is deprecated.</i> Use <code>Qt::SystemLocaleShortDate</code> instead (or <code>Qt::SystemLocaleLongDate</code> if you want long dates).
<code>Qt::RFC2822Date</code>	8	<a href="#">RFC 2822</a> , <a href="#">RFC 850</a> and <a href="#">RFC 1036</a> format: either [ddd,] dd MMM yyyy hh:mm[:ss] +/-TZ or ddd MMM dd yyyy hh:mm[:ss] +/-TZ for combined dates and times.

## Section 02 기초

### □ 날짜와 시간 표시하기

- 시간 표시하기(QTime)

- QTime 클래스를 사용해서 현재 시간을 출력할 수 있다.

```
>>> from PyQt6.QtCore import QTime
>>> time = QTime.currentTime()
>>> print(time.toString())
```

- currentTime() 메서드는 현재 시간을 반환한다.
- toString() 메서드는 현재 시간을 문자열로 반환한다.
- 결과는 아래와 같다.

```
>>> >>> from PyQt6.QtCore import QTime
=> >>> time = QTime.currentTime()
>>> print(time.toString())
00:23:47
```



## Section 02 기초

### □ 날짜와 시간 표시하기

#### • 시간 형식 설정하기

```
>>> from PyQt6.QtCore import QDateTime, Qt
>>> datetime = QDateTime.currentDateTime()
>>> print(datetime.toString('d.M.yy hh:mm:ss'))
>>> print(datetime.toString('dd.MM.yyyy, hh:mm:ss'))
```

- 'h'는 시간(hour), 'm'은 분(minute), 's'는 초(second), 그리고 'z'는 1000분의 1초를 나타낸다.
- 또한 날짜에서와 마찬가지로 Qt.DateFormat.DefaultLocaleShortDate 또는 Qt.DateFormat.DefaultLocaleLongDate 등으로 시간의 형식을 설정할 수 있다.

# Section 02 기초

## □ 날짜와 시간 표시하기

### • 시간 형식 설정하기

➤ 자세한 내용은 아래의 표 또는 공식 문서를 참고하면 된다.

Expression	Output
h	the hour without a leading zero (0 to 23 or 1 to 12 if AM/PM display)
hh	the hour with a leading zero (00 to 23 or 01 to 12 if AM/PM display)
H	the hour without a leading zero (0 to 23, even with AM/PM display)
HH	the hour with a leading zero (00 to 23, even with AM/PM display)
m	the minute without a leading zero (0 to 59)
mm	the minute with a leading zero (00 to 59)
s	the whole second, without any leading zero (0 to 59)
ss	the whole second, with a leading zero where applicable (00 to 59)
z	the fractional part of the second, to go after a decimal point, without trailing zeroes (0 to 999). Thus "s.z" reports the seconds to full available (millisecond) precision without trailing zeroes.
zzz	the fractional part of the second, to millisecond precision, including trailing zeroes where applicable (000 to 999).
AP or A	use AM/PM display. <i>A/AP</i> will be replaced by either <code>QLocale::amText()</code> or <code>QLocale::pmText()</code> .
ap or a	use am/pm display. <i>a/ap</i> will be replaced by a lower-case version of <code>QLocale::amText()</code> or <code>QLocale::pmText()</code> .
t	the timezone (for example "CEST")

## Section 02 기초

---

### □ 날짜와 시간 표시하기

- 날짜와 시간 표시하기(QDateTime)

- QDateTime 클래스를 사용해서 현재 날짜와 시간을 함께 출력할 수 있다.

```
>>> from PyQt6.QtCore import QDateTime
>>> datetime = QDateTime.currentDateTime()
>>> print(datetime.toString())
```

- currentDateTime() 메서드는 현재의 날짜와 시간을 반환한다.
- toString() 메서드는 날짜와 시간을 문자열 형태로 반환한다.

## Section 02 기초

### □ 날짜와 시간 표시하기

- 날짜와 시간 표시하기(QDateTime)

- 위의 예제에서와 마찬가지로 날짜에 대해 'd', 'M', 'y', 시간에 대해 'h', 'm', 's' 등을 사용해서 날짜와 시간이 표시되는 형식을 설정할 수 있다.
- 또한 Qt.DefaultLocaleLongDate 또는 Qt.DefaultLocaleShortDate를 입력할 수 있다.

```
>>> from PyQt6.QtCore import QDateTime, Qt
>>> datetime = QDateTime.currentDateTime()
>>> print(datetime.toString('d.M.yy hh:mm:ss'))
>>> print(datetime.toString('dd.MM.yyyy, hh:mm:ss'))
```

## Section 02 기초

### □ 날짜와 시간 표시하기

- 상태표시줄에 날짜 표시하기

➤ ex\_1\_9.py

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QMainWindow
3  from PyQt6.QtCore import QDate, Qt
4  from time import strftime, localtime
5
6  class MyApp(QMainWindow):
7      def __init__(self):
8          super().__init__()
9          self.date = QDate.currentDate()
10         self.initUI()
11
12     def initUI(self):
13         self.statusBar().showMessage(self.date.toString(strftime("%Y년 %m월 %d일",
14                                                                     localtime())))
```

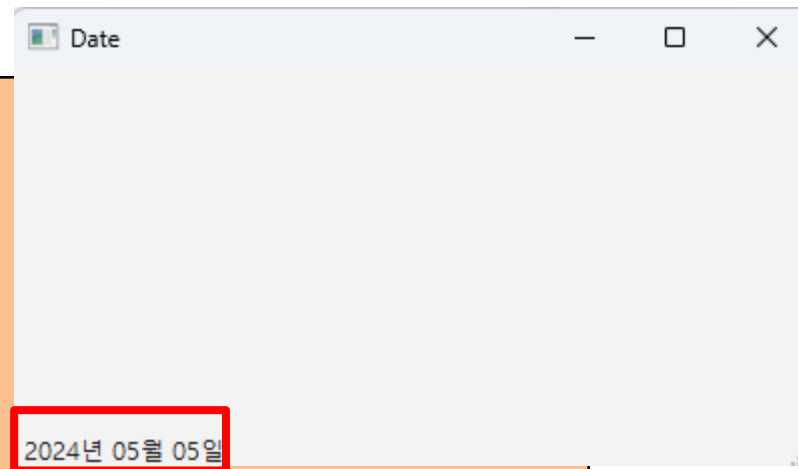
## Section 02 기초

### □ 날짜와 시간 표시하기

- 상태표시줄에 날짜 표시하기

➤ ex\_1\_9.py

```
14
15     self.setWindowTitle('Date')
16     self.setGeometry(300, 300, 400, 200)
17     self.show()
18
19 if __name__ == '__main__':
20     app = QApplication(sys.argv)
21     ex = MyApp()
22     app.exec()
```



➤ currentDate() 메서드를 통해 현재 날짜를 얻고 showMessage() 메서드로 상태표시줄에 현재 날짜를 표시했다.

## Section 02 기초

### □스타일 꾸미기

- `setStyleSheet()`을 이용하면 어플리케이션 안의 다양한 구성 요소들의 스타일을 자유롭게 꾸밀 수 있다.
- `ex_1_10.py`

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QVBoxLayout
3
4  class MyApp(QWidget):
5      def __init__(self):
6          super().__init__()
7          self.initUI()
8
9      def initUI(self):
10         lbl_red = QLabel('Red')
11         lbl_green = QLabel('Green')
12         lbl_blue = QLabel('Blue')
```

## Section 02 기초

### □스타일 꾸미기

- ex\_1\_10.py

```
13
14     lbl_red.setStyleSheet("color: red;"
15                             "border-style: solid;"
16                             "border-width: 2px;"
17                             "border-color: #FA8072;"
18                             "border-radius: 3px")
19     lbl_green.setStyleSheet("color: green;"
20                             "background-color: #7FFFD4")
21     lbl_blue.setStyleSheet("color: blue;"
22                             "background-color: #87CEFA;"
23                             "border-style: dashed;"
24                             "border-width: 3px;"
25                             "border-color: #1E90FF")
26
```

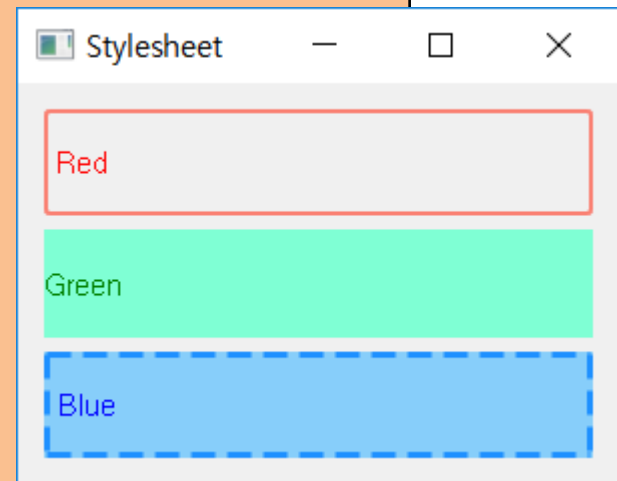


# Section 02 기초

## □스타일 꾸미기

- ex\_1\_10.py

```
27     vbox = QVBoxLayout()
28     vbox.addWidget(lbl_red)
29     vbox.addWidget(lbl_green)
30     vbox.addWidget(lbl_blue)
31
32     self.setLayout(vbox)
33
34     self.setWindowTitle('Stylesheet')
35     self.setGeometry(300, 300, 300, 200)
36     self.show()
37
38 if __name__ == '__main__':
39     app = QApplication(sys.argv)
40     ex = MyApp()
41     app.exec()
```



## Section 02 기초

### □스타일 꾸미기

```
lbl_red.setStyleSheet("color: red;"  
                        "border-style: solid;"  
                        "border-width: 2px;"  
                        "border-color: #FA8072;"  
                        "border-radius: 3px")
```

- QLabel 클래스를 이용해서 세 개의 라벨 위젯을 만든다.
- 라벨 텍스트는 각각 'Red', 'Green', 'Blue'로 설정한다.

```
lbl_red.setStyleSheet("color: red;"  
                        "border-style: solid;"  
                        "border-width: 2px;"  
                        "border-color: #FA8072;"  
                        "border-radius: 3px")
```

- **setStyleSheet() 메서드를 이용해서 글자색을 빨간색(red)으로, 경계선을 실선(solid)으로, 경계선 두께를 2px로, 경계선 색을 #FA8072로, 경계선의 모서리를 3px만큼 둥글게 설정한다.**

## Section 02 기초

### □스타일 꾸미기

```
lbl_green.setStyleSheet("color: green;"  
                        "background-color: #7FFFD4")
```

- 마찬가지로, lbl\_green 라벨의 글자색을 녹색(green)으로, 배경색을 #7FFFD4로 설정합니다.

```
lbl_blue.setStyleSheet("color: blue;"  
                      "background-color: #87CEFA;"  
                      "border-style: dashed;"  
                      "border-width: 3px;"  
                      "border-color: #1E90FF")
```

- lbl\_blue 라벨의 글자색을 파란색(blue)으로, 배경색을 #87CEFA으로, 경계선을 대쉬 스타일로, 경계선 두께를 3px로, 경계선 색을 #1E90FF으로 설정한다.

## Section 02 기초

### □스타일 꾸미기

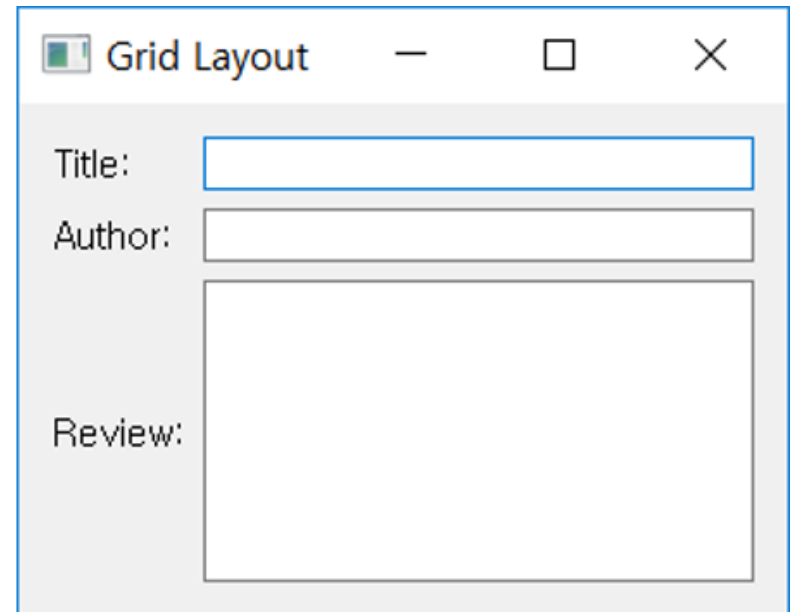
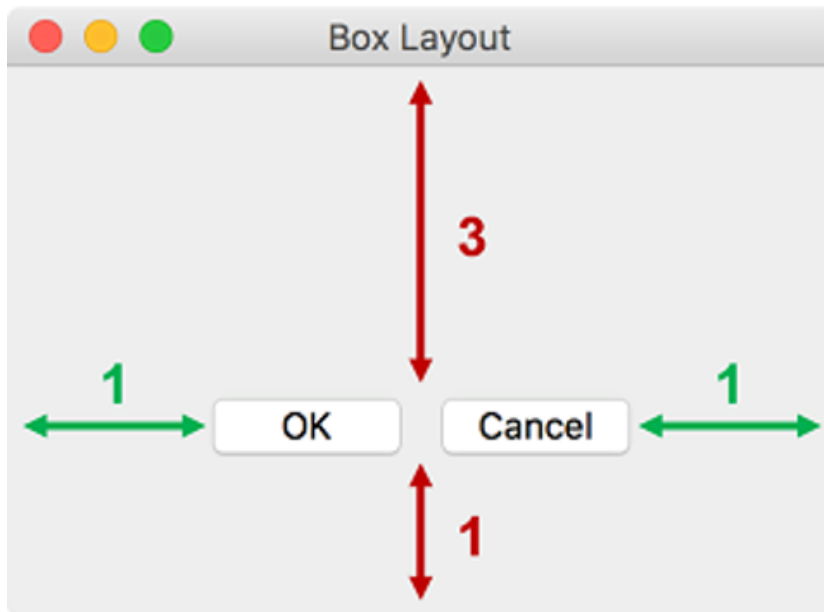
```
vbox = QVBoxLayout()  
vbox.addWidget(lbl_red)  
vbox.addWidget(lbl_green)  
vbox.addWidget(lbl_blue)  
  
self.setLayout(vbox)
```

- 수직 박스 레이아웃(QVBoxLayout())을 이용해서 세 개의 라벨 위젯을 수직으로 배치한다.
- 수직 박스 레이아웃에 대한 설명은 박스 레이아웃 페이지 설명을 참고하면 된다.
- 더 다양한 스타일 항목은 스타일 시트 Reference 페이지 설명을 참고하면 된다.

## Section 03 레이아웃

### □개요

- 레이아웃 (Layout)은 어플리케이션 창에 위젯들을 배치하는 방식이다.
- 레이아웃 관리는 GUI 프로그래밍에서 매우 중요한 요소이다.
- PyQt5의 위젯들을 배치하는 방식에는 절대적 배치, 박스 레이아웃, 그리드 레이아웃 방식이 있다.



## Section 03 레이아웃

---

### □절대적 배치

- **절대적 배치**(Absolute positioning) 방식은 각 위젯의 위치와 크기를 픽셀 단위로 설정해서 배치한다.
- **절대 배치 방식을 사용할 때는 다음의 제약을 이해하고 있어야 한다.**
  - 창의 크기를 조절해도 위젯의 크기와 위치는 변하지 않는다.
  - 다양한 플랫폼에서 어플리케이션이 다르게 보일 수 있다.
  - 어플리케이션의 폰트를 바꾸면 레이아웃이 망가질 수 있다.
  - 레이아웃을 바꾸고 싶다면 완전히 새로 고쳐야 하며, 이는 매우 번거롭다.
- **두 개의 라벨과 두 개의 푸시버튼 위젯을 절대적 배치 방식으로 배치해 보자.**

# Section 03 레이아웃

## □절대적 배치

- ex\_2\_1.py

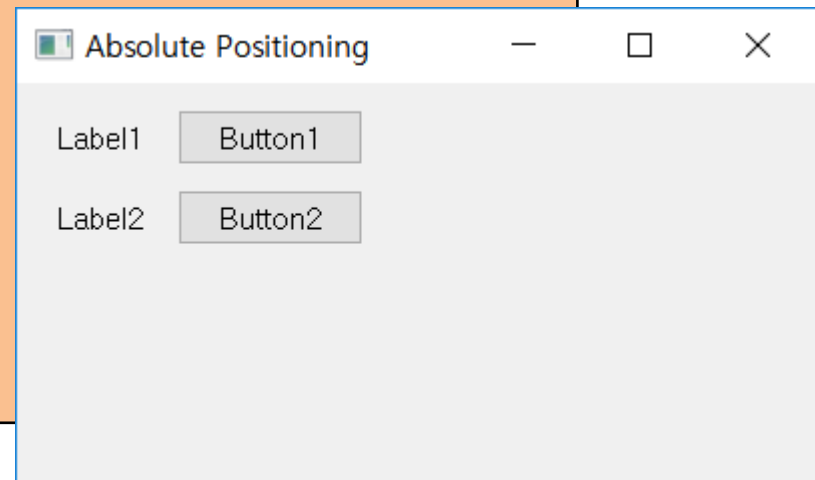
```
1 import sys
2 from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QPushButton
3
4 class MyApp(QWidget):
5     def __init__(self):
6         super().__init__()
7         self.initUI()
8
9     def initUI(self):
10         label1 = QLabel('Label1', self)
11         label1.move(20, 20)
12         label2 = QLabel('Label2', self)
13         label2.move(20, 60)
14
```

# Section 03 레이아웃

## □절대적 배치

- ex\_2\_1.py

```
15     btn1 = QPushButton('Button1', self)
16     btn1.move(80, 13)
17     btn2 = QPushButton('Button2', self)
18     btn2.move(80, 53)
19
20     self.setWindowTitle('Absolute Positioning')
21     self.setGeometry(300, 300, 400, 200)
22     self.show()
23
24 if __name__ == '__main__':
25     app = QApplication(sys.argv)
26     ex = MyApp()
27     app.exec()
```





## Section 03 레이아웃

### □절대적 배치

```
label1 = QLabel('Label1', self)
label1.move(20, 20)
```

- 라벨을 하나 만들고,  $x=20$ ,  $y=20$ 에 위치하도록 `move` 메서드를 사용.
- 좌표계는 왼쪽 상단 모서리에서 시작한다.
- $x$  좌표는 왼쪽에서 오른쪽으로 갈수록 커지고,  $y$  좌표는 위에서 아래로 갈수록 커진다.

```
btn1 = QPushButton('Button1', self)
btn1.move(80, 13)
```

- 푸시버튼을 하나 만들고,  $x=80$ ,  $y=13$ 에 위치하도록 이동시킨다.

## Section 03 레이아웃

---

### □ 박스 레이아웃

- 박스 레이아웃 클래스를 이용하면 훨씬 유연하고 실용적인 레이아웃을 할 수 있다. (QBoxLayout 공식 문서 참고)
- QHBoxLayout, QVBoxLayout은 여러 위젯을 수평으로 정렬하는 레이아웃 클래스이다.
- QHBoxLayout, QVBoxLayout 생성자는 수평, 수직의 박스를 하나 만드는데, 다른 레이아웃 박스를 넣을 수도 있고 위젯을 배치할 수도 있다.
- 예제 코드에서 위젯의 가운데 아래 부분에 두 개의 버튼을 배치하기 위해 수평, 수직의 박스를 하나씩 사용한다.
- 필요한 공간을 만들기 위해 `addStretch()` 메서드를 사용하고, 'stretch factor'를 조절해 보자.

# Section 03 레이아웃

## □ 박스 레이아웃

- ex\_2\_2.py

```
1 import sys
2 from PyQt6.QtWidgets import (QApplication, QWidget, QPushButton,
3                               QHBoxLayout, QVBoxLayout)
4
5 class MyApp(QWidget):
6     def __init__(self):
7         super().__init__()
8         self.initUI()
9
10    def initUI(self):
11        okButton = QPushButton('OK')
12        cancelButton = QPushButton('Cancel')
13
14        hbox = QHBoxLayout()
15        hbox.addStretch(1)
```

## Section 03 레이아웃

### □ 박스 레이아웃

- ex\_2\_2.py

```
15     hbox.addWidget(okButton)
16     hbox.addWidget(cancelButton)
17     hbox.addStretch(1)
18
19     vbox = QVBoxLayout()
20     vbox.addStretch(3)
21     vbox.addLayout(hbox)
22     vbox.addStretch(1)
23
24     self.setLayout(vbox)
25
26     self.setWindowTitle('Box Layout')
27     self.setGeometry(300, 300, 300, 200)
28     self.show()
```

## Section 03 레이아웃

### □ 박스 레이아웃

- ex\_2\_2.py

```
29
30     if __name__ == '__main__':
31         app = QApplication(sys.argv)
32         ex = MyApp()
33         app.exec()
```

- 창의 가운데 아래에 두 개의 버튼을 배치시킨다.
- 두 개의 버튼은 창의 크기를 변화시켜도 같은 자리에 위치한다.

## Section 03 레이아웃

### □ 박스 레이아웃

```
okButton = QPushButton('OK')  
cancelButton = QPushButton('Cancel')
```

- 두 개의 버튼을 생성

```
hbox = QHBoxLayout()  
hbox.addStretch(1)  
hbox.addWidget(okButton)  
hbox.addWidget(cancelButton)  
hbox.addStretch(1)
```

- 수평 박스를 하나 만들고, 두 개의 버튼과 양 쪽에 빈 공간을 추가한다.
- 이 `addStretch()` 메서드는 신축성있는 빈 공간을 제공한다.
- 두 버튼 양쪽의 stretch factor가 1로 같기 때문에 이 두 빈 공간의 크기는 창의 크기가 변화해도 항상 같다.

## Section 03 레이아웃

### □ 박스 레이아웃

```
vbox = QVBoxLayout()  
vbox.addStretch(3)  
vbox.addLayout(hbox)  
vbox.addStretch(1)
```

- 다음으로 수평 박스(hbox)를 수직 박스(vbox)에 넣는다.
- 수직 박스의 stretch factor는 수평 박스를 아래쪽으로 밀어내서 두 개의 버튼을 창의 아래쪽에 위치시킨다.
- 이 때에도 수평 박스 위와 아래의 빈 공간의 크기는 항상 3:1을 유지한다.
- Stretch factor를 다양하게 바꿔보면, 의미를 잘 이해할 수 있다.

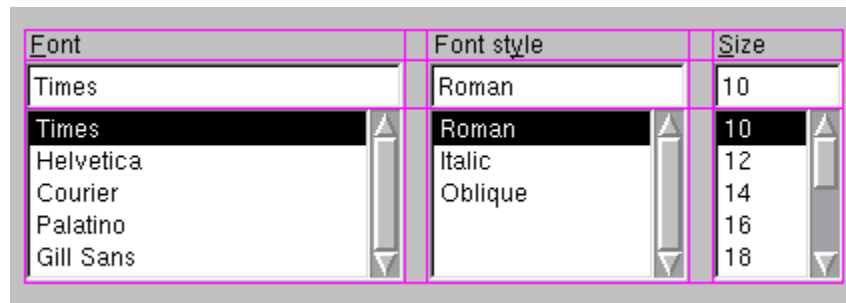
```
self.setLayout(vbox)
```

- 최종적으로 수직 박스를 창의 메인 레이아웃으로 설정한다.

## Section 03 레이아웃

### □그리드 레이아웃

- 가장 일반적인 레이아웃 클래스는 '그리드 레이아웃(grid layout) ' 이다.
- 이 레이아웃 클래스는 위젯의 공간을 **행 (row)**과 **열 (column)**로 구분한다.
- 그리드 레이아웃을 생성하기 위해 **QGridLayout** 클래스를 사용한다.



- 위 예시 다이얼로그의 경우, 세 개의 **행(Row)**과 다섯 개의 **열(Column)**로 구분되어 있고, 필요한 위치에 위젯을 배치했다.



# Section 03 레이아웃

## □그리드 레이아웃

- ex\_2\_3.py

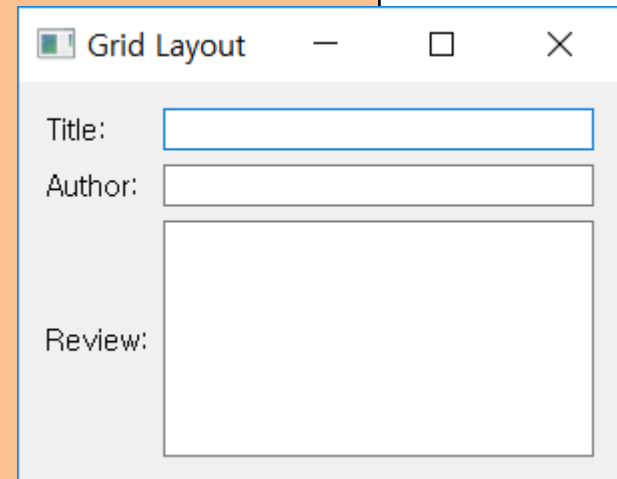
```
1 import sys
2 from PyQt6.QtWidgets import (QApplication, QWidget, QGridLayout, QLabel,
3                               QLineEdit, QTextEdit)
4
5 class MyApp(QWidget):
6     def __init__(self):
7         super().__init__()
8         self.initUI()
9
10    def initUI(self):
11        grid = QGridLayout()
12        self.setLayout(grid)
13
14        grid.addWidget(QLabel('Title:'), 0, 0)
15        grid.addWidget(QLabel('Author:'), 1, 0)
```

# Section 03 레이아웃

## □그리드 레이아웃

- ex\_2\_3.py

```
15     grid.addWidget(QLabel('Review:'), 2, 0)
16
17     grid.addWidget(QLineEdit(), 0, 1)
18     grid.addWidget(QLineEdit(), 1, 1)
19     grid.addWidget(QTextEdit(), 2, 1)
20
21     self.setWindowTitle('QGridLayout')
22     self.setGeometry(300, 300, 300, 200)
23     self.show()
24
25 if __name__ == '__main__':
26     app = QApplication(sys.argv)
27     ex = MyApp()
28     app.exec()
```



## Section 03 레이아웃

### □그리드 레이아웃

- 위 예제는 세 개의 라벨, 두 개의 라인 에디터, 하나의 텍스트 에디터를 그리드 형태로 배치했다.

```
grid = QGridLayout()  
self.setLayout(grid)
```

- QGridLayout을 만들고, 어플리케이션 창의 레이아웃으로 설정한다.

```
grid.addWidget(QLabel('Title:'), 0, 0)  
grid.addWidget(QLabel('Author:'), 1, 0)  
grid.addWidget(QLabel('Review:'), 2, 0)
```

- addWidget() 메서드의 첫 번째 위젯은 추가할 위젯, 두, 세 번째 위젯은 각각 행과 열 번호를 입력한다.
- 세 개의 라벨을 첫 번째 열에 수직으로 배치한다.

## Section 03 레이아웃

---

### □그리드 레이아웃

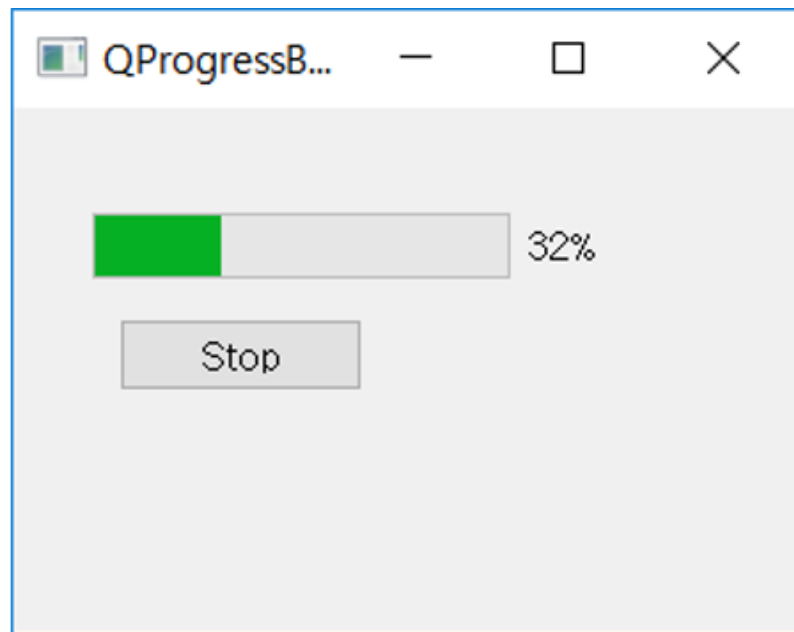
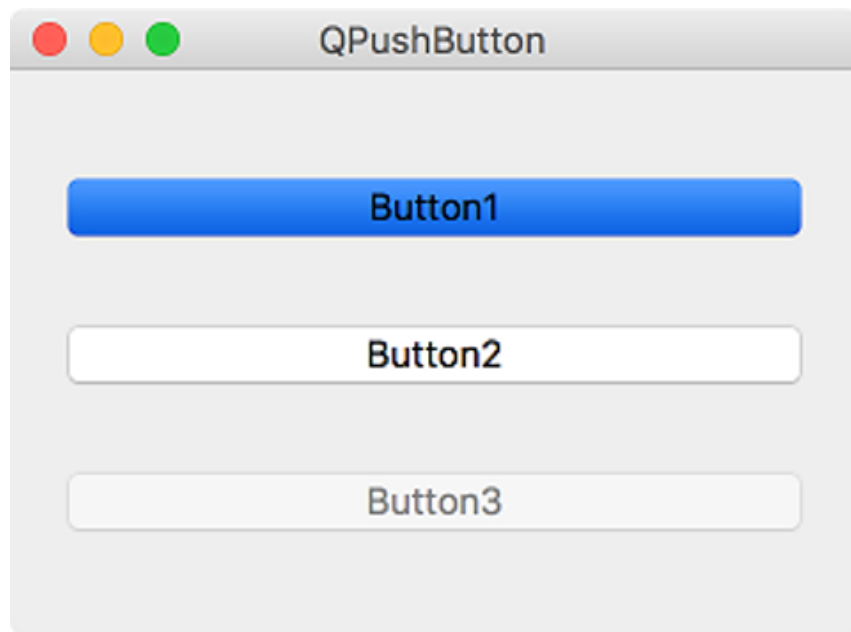
```
grid.addWidget(QTextEdit(), 2, 1)
```

- QTextEdit() 위젯은 QLineEdit() 위젯과 달리 여러 줄의 텍스트를 수정할 수 있는 위젯이다.
- 세 번째 행, 두 번째 열에 배치한다.

## Section 04 위젯

### □개요

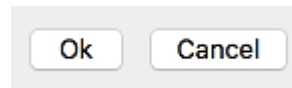
- 위젯은 어플리케이션을 이루는 기본적인 구성 요소이다.
- PyQt 툴킷은 간단하게 사용할 수 있는 다양한 위젯을 제공한다.
- 이제 어플리케이션에서 흔하게 사용되는 여러 위젯들을 예제를 통해 알아보자.



## Section 04 위젯

### □QPushButton

- **푸시 버튼(push button) 또는 명령 버튼(command button)**은 사용자가 프로그램에 명령을 내려서 어떤 동작을 하도록 할 때 사용되는 버튼이며, GUI 프로그래밍에서 가장 흔하게 사용되고 중요한 위젯이다.



- 자주 쓰이는 메서드

메서드	설명
setCheckable()	True 설정 시, 누른 상태와 그렇지 않은 상태를 구분한다.
toggle()	상태를 바꾼다.
setIcon()	버튼의 아이콘을 설정한다.
setEnabled()	False 설정 시, 버튼을 사용할 수 없다.
isChecked()	버튼의 선택 여부를 반환한다.
setText()	버튼에 표시될 텍스트를 설정한다.
text()	버튼에 표시된 텍스트를 반환한다.

## Section 04 위젯

### □QPushButton

- 자주 쓰이는 시그널

시그널	설명
clicked()	버튼을 클릭할 때 발생한다.
pressed()	버튼이 눌렸을 때 발생한다.
released()	버튼을 눌렀다 떼 때 발생한다.
toggled()	버튼의 상태가 바뀔 때 발생한다.

- 이제 QPushButton 클래스를 이용해서 푸시 버튼을 만들어보자.

# Section 04 위젯

## □ QPushButton

- ex\_4\_1.py

```
1 import sys
2 from PyQt6.QtWidgets import QApplication, QWidget, QPushButton, QVBoxLayout
3
4 class MyApp(QWidget):
5     def __init__(self):
6         super().__init__()
7         self.initUI()
8
9     def initUI(self):
10         btn1 = QPushButton('&Button1', self)
11         btn1.setCheckable(True)
12         btn1.toggle()
13
14         btn2 = QPushButton(self)
```



# Section 04 위젯

## □ QPushButton

- ex\_4\_1.py

```
15         btn2.setText('Button&2')
16
17         btn3 = QPushButton('Button3', self)
18         btn3.setEnabled(False)
19
20         vbox = QVBoxLayout()
21         vbox.addWidget(btn1)
22         vbox.addWidget(btn2)
23         vbox.addWidget(btn3)
24
25         self.setLayout(vbox)
26         self.setWindowTitle('QPushButton')
27         self.setGeometry(300, 300, 300, 200)
28         self.show()
```

# Section 04 위젯

---

## □ QPushButton

- ex\_4\_1.py

```
29  
30     if __name__ == '__main__':  
31         app = QApplication(sys.argv)  
32         ex = MyApp()  
33         app.exec()
```

## Section 04 위젯

### □ QPushButton

```
btn1 = QPushButton('&Button1', self)
btn1.setCheckable(True)
btn1.toggle()
```

- QPushButton 클래스로 푸시 버튼을 하나 만든다.
- 첫 번째 파라미터로는 버튼에 나타날 텍스트, 두 번째는 버튼이 속할 부모 클래스를 지정해준다.
- 버튼에 단축키(shortcut)를 지정하고 싶으면 아래와 같이 해당 문자 앞에 ampersand('&')를 넣어주면 된다.
- 이 버튼의 단축키는 ' Alt+b ' 가 된다.
- setCheckable()을 True로 설정해주면, 선택되거나 선택되지 않은 상태를 유지할 수 있게 된다.
- toggle() 메서드를 호출하면 버튼의 상태가 바뀌게 된다.
- 따라서 이 버튼은 프로그램이 시작될 때 선택되어 있다.

## Section 04 위젯

---

### □ QPushButton

```
btn2 = QPushButton(self)
btn2.setText('Button&2')
```

- setText() 메서드로도 버튼에 표시될 텍스트를 지정할 수 있다.
- 또한 이 버튼의 단축키는 'Alt+2 ' 가 된다.

```
btn3 = QPushButton('Button3', self)
btn3.setEnabled(False)
```

- setEnabled()를 False로 설정하면, 버튼을 사용할 수 없게 된다.

## Section 04 위젯

---

### □QLabel

- QLabel 위젯은 텍스트 또는 이미지 라벨을 만들 때 사용된다.
- 사용자와 어떤 상호작용을 제공하지는 않는다. (QLabel 공식 문서 참고)
- 라벨은 기본적으로 수평 방향으로서는 왼쪽, 수직 방향으로서는 가운데 정렬이지만 `setAlignment()` 메서드를 통해 조절할 수 있다.
- 라벨을 하나 만들고, 라벨의 스타일과 관련된 몇 개의 메서드들을 사용해 보자.

# Section 04 위젯

## □ QLabel

- ex\_4\_2.py

```
1 import sys
2 from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QVBoxLayout
3 from PyQt6.QtCore import Qt
4
5 class MyApp(QWidget):
6     def __init__(self):
7         super().__init__()
8         self.initUI()
9
10    def initUI(self):
11        label1 = QLabel('First Label', self)
12        label1.setAlignment(Qt.AlignmentFlag.AlignVCenter)
13
14        label2 = QLabel('Second Label', self)
15        label2.setAlignment(Qt.AlignmentFlag.AlignVCenter)
```

# Section 04 위젯

## □ QLabel

- ex\_5\_2.py

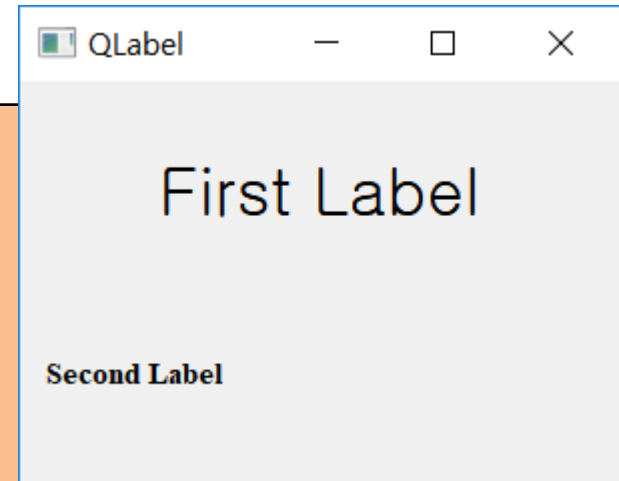
```
16
17     font1 = label1.font()
18     font1.setPointSize(20)
19
20     font2 = label2.font()
21     font2.setFamily('Times New Roman')
22     font2.setBold(True)
23
24     label1.setFont(font1)
25     label2.setFont(font2)
26
27     layout = QVBoxLayout()
28     layout.addWidget(label1)
29     layout.addWidget(label2)
30
```

# Section 04 위젯

## □ QLabel

- ex\_5\_2.py

```
31         self.setLayout(layout)
32
33         self.setWindowTitle('QLabel')
34         self.setGeometry(300, 300, 300, 200)
35         self.show()
36
37     if __name__ == '__main__':
38         app = QApplication(sys.argv)
39         ex = MyApp()
40         app.exec()
```



- 이 예제는 두 개의 라벨을 수직 박스에 넣어서 표시했다.



## Section 04 위젯

### □ QLabel

```
label1 = QLabel('First Label', self)
label1.setAlignment(Qt.AlignmentFlag.AlignVCenter)
```

- QLabel 위젯을 하나 생성한다.
- 생성자에 라벨 텍스트와 부모 위젯을 입력한다.
- `setAlignment()` 메서드로 라벨의 배치를 설정할 수 있다.
- `Qt.AlignmentFlag.AlignVCenter`로 설정해주면 수평, 수직 방향 모두 가운데 위치하게 된다.

```
font1 = label1.font()
font1.setPointSize(20)
```

- 라벨에 사용될 폰트를 하나 생성한다.
- `setPointSize()` 메서드로 폰트의 크기를 설정한다.

## Section 04 위젯

### □QLabel

```
label2 = QLabel('Second Label', self)
label2.setAlignment(Qt.AlignmentFlag.AlignVCenter)
```

- 두번째 라벨을 만들고, 이번에는 수직 방향으로만 가운데 (Qt.AlignmentFlag.AlignVCenter)로 설정한다.
- 수평 방향으로 가운데로 설정하려면 Qt.AlignmentFlag.AlignHCenter 를 입력해주면 된다.

```
font2 = label2.font()
font2.setFamily('Times New Roman')
font2.setBold(True)
```

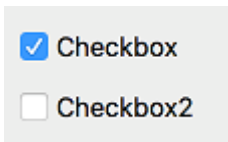
- 두번째 라벨에 설정할 폰트를 하나 만들고, setFamily() 메서드로 폰트의 종류를 'Times New Roman'으로 설정한다.
- setBold(True)로 폰트를 진하게 설정한다.

이번에는 폰트의 크기를 설정하지 않았기 때문에 디폴트 크기인 13으로 설정된다.

## Section 04 위젯

### □QCheckBox

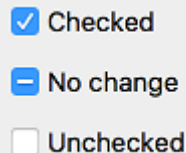
- QCheckBox 위젯은 on(체크됨)/off(체크안됨)의 두 상태를 갖는 버튼을 제공한다.
- 이 위젯은 하나의 텍스트 라벨과 함께 체크 박스를 제공한다. (QCheckBox 공식 문서 참고)
- 체크 박스가 선택되거나 해제될 때, stateChanged() 시그널을 발생시킨다.
- 체크 박스의 상태가 변할 때마다 어떠한 동작을 발생시키고 싶을 때, 이 시그널을 특정 슬롯에 연결할 수 있다.
- 또한 체크 박스의 선택 여부를 확인하기 위해서, isChecked() 메서드를 사용할 수 있다.
- 선택 여부에 따라 boolean 값을 반환한다.



## Section 04 위젯

### □QCheckBox

- 일반적인 체크 박스는 선택/해제 상태만을 갖지만, `setTristate()` 메서드를 사용하면 '변경 없음(no change)' 상태를 가질 수 있다.
- 이 체크 박스는 사용자에게 선택하거나 선택하지 않을 옵션을 줄 때 유용하다.
- 세 가지 상태를 갖는 체크 박스의 상태를 얻기 위해서는 `checkState()` 메서드를 사용한다.
- 선택/변경 없음/해제 여부에 따라 각각 2/1/0 값을 반환한다.
- `QButtonGroup` 클래스를 사용하면 여러 개의 버튼을 묶어서 exclusive/non-exclusive 버튼 그룹을 만들 수 있다.
- exclusive 버튼 그룹은 여러 개 중 하나의 버튼만 선택할 수 있다.  
(`QButtonGroup` 공식 문서 참고)



## Section 04 위젯

### □QCheckBox

- 자주 쓰이는 메서드

메서드	설명
text()	체크 박스의 라벨 텍스트를 반환한다.
setText()	체크 박스의 라벨 텍스트를 설정한다.
isChecked()	체크 박스의 상태를 반환한다. (True/False)
checkState()	체크 박스의 상태를 반환한다. (2/1/0)
toggle()	체크 박스의 상태를 변경한다.

- 자주 쓰이는 시그널

시그널	설명
pressed()	체크 박스를 누를 때 신호를 발생시킨다.
released()	체크 박스에서 땔 때 신호를 발생시킨다.
clicked()	체크 박스를 클릭할 때 신호를 발생시킨다.
stateChanged()	체크 박스의 상태가 바뀔 때 신호를 발생시킨다.

# Section 04 위젯

## ❑QCheckBox

- ex\_4\_3.py

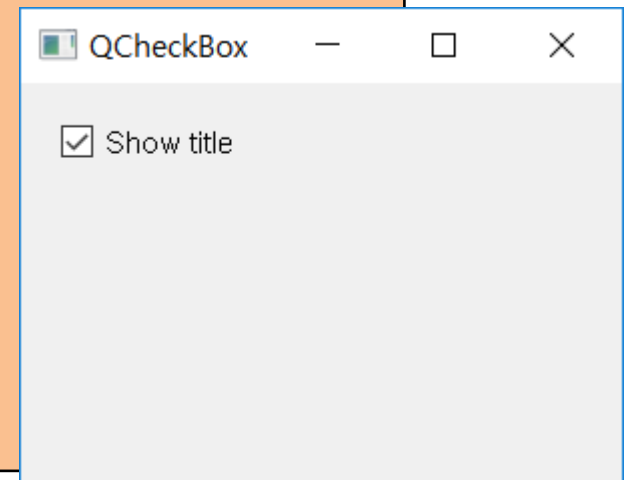
```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget, QCheckBox
3  from PyQt6.QtCore import Qt
4
5  class MyApp(QWidget):
6      def __init__(self):
7          super().__init__()
8          self.initUI()
9
10     def initUI(self):
11         cb = QCheckBox('Show title', self)
12         cb.move(20, 20)
13         cb.toggle()
14         cb.stateChanged.connect(self.changeTitle)
15
```

# Section 04 위젯

## ❑ QCheckBox

- ex\_4\_3.py

```
15     self.setWindowTitle('QCheckBox')
16     self.setGeometry(300, 300, 300, 200)
17     self.show()
18
19     def changeTitle(self, state):
20         if Qt.CheckState(state) == Qt.CheckState.Checked:
21             self.setWindowTitle('QCheckBox')
22         else:
23             self.setWindowTitle(' ')
24
25 if __name__ == '__main__':
26     app = QApplication(sys.argv)
27     ex = MyApp()
28     app.exec()
```



## Section 04 위젯

---

### ❑QCheckBox

- 위 예제는 체크박스를 하나 만들고, 체크가 되어 있을 때 타이틀바에 ' QCheckBox ' 글자가 나타나도록 했다.
- 체크박스의 상태에 따라 타이틀이 나타나고 사라진다.

```
cb = QCheckBox('Show title', self)
```

- 'Show title'이라는 텍스트 라벨을 갖는 체크박스를 하나 생성한다.

```
cb.toggle()
```

- 체크박스는 디폴트로 체크가 되어있지 않은 off 상태로 나타나기 때문에 on 상태로 바꾸기 위해 toggle() 메서드를 사용했다.



## Section 04 위젯

### ❑QCheckBox

```
cb.stateChanged.connect(self.changeTitle)
```

- **체크박스의 상태가 바뀔 때 발생하는 시그널 (stateChanged)을 우리가 정의한 changeTitle() 메서드에 연결한다.**

```
def changeTitle(self, state):
```

```
    if Qt.CheckState(state) == Qt.CheckState.Checked:
```

```
        self.setWindowTitle('QCheckBox')
```

```
    else:
```

```
        self.setWindowTitle('')
```

- **체크박스의 상태 (state)가 changeTitle() 메서드의 매개변수로 주어진다.**
- **체크가 되어있으면 타이틀을 'QCheckBox'로, 그렇지 않으면 빈 문자열로 나타난다.**

## Section 04 위젯

---

### □QRadioButton

- RadioButton 위젯은 사용자가 선택할 수 있는 버튼을 만들 때 사용한다.
- 이 버튼에도 체크 박스와 마찬가지로 텍스트 라벨이 하나 포함된다.  
(QRadioButton 공식 문서 참고)
- 라디오 버튼은 일반적으로 사용자에게 여러 개 중 하나의 옵션을 선택하도록 할 때 사용된다.
- 그래서 한 위젯 안에 여러 라디오 버튼은 기본적으로 `autoExclusive`로 설정되어 있다.
- 하나의 버튼을 선택하면 나머지 버튼들은 선택 해제가 된다.
- 한 번에 여러 버튼을 선택할 수 있도록 하려면 `setAutoExclusive()` 메서드에 `False`를 입력해주면 된다.
- 또한 한 위젯 안에 여러 개의 `exclusive` 버튼 그룹을 배치하고 싶다면 `QButtonGroup()` 메서드를 사용할 수 있다. (`QButtonGroup` 공식 문서 참고)

## Section 04 위젯

### □QRadioButton

- 체크 박스와 마찬가지로 버튼의 상태가 바뀔 때, toggled() 시그널이 발생한다.
- 또한 특정 버튼의 상태를 가져오고 싶을 때, isChecked() 메서드를 사용할 수 있다.
- 자주 쓰이는 메서드

메서드	설명
text()	버튼의 텍스트를 반환한다.
setText()	라벨에 들어갈 텍스트를 설정한다.
setChecked()	버튼의 선택 여부를 설정한다.
isChecked()	버튼의 선택 여부를 반환한다.
toggle()	버튼의 상태를 변경한다.

## Section 04 위젯

---

### □QRadioButton

- 자주 쓰이는 시그널

시그널	설명
pressed()	버튼을 누를 때 신호를 발생시킨다.
released()	버튼에서 떼를 때 신호를 발생시킨다.
clicked()	버튼을 클릭할 때 신호를 발생시킨다.
toggled()	버튼의 상태가 바뀔 때 신호를 발생시킨다.

# Section 04 위젯

## ❑QRadioButton

- ex\_4\_4.py

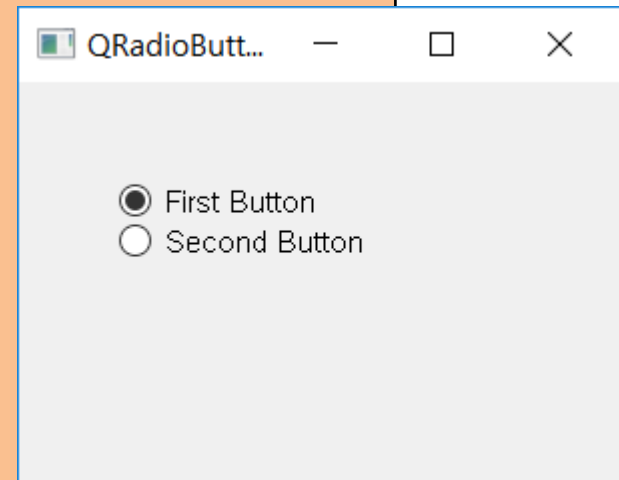
```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget, QRadioButton
3
4  class MyApp(QWidget):
5      def __init__(self):
6          super().__init__()
7          self.initUI()
8
9      def initUI(self):
10         rbtn1 = QRadioButton('First Button', self)
11         rbtn1.move(50, 50)
12         rbtn1.setChecked(True)
13
14         rbtn2 = QRadioButton(self)
15         rbtn2.move(50, 70)
```

## Section 04 위젯

### ❑ QRadioButton

- ex\_4\_4.py

```
16         rbtn2.setText('Second Button')
17
18         self.setGeometry(300, 300, 300, 200)
19         self.setWindowTitle('QRadioButton')
20         self.show()
21
22     if __name__ == '__main__':
23         app = QApplication(sys.argv)
24         ex = MyApp()
25         app.exec()
```



- 이 예제는 두 개의 라디오버튼을 만들었다.

## Section 04 위젯

---

### □QRadioButton

```
rbtn1 = QRadioButton('First Button', self)
```

- QRadioButton을 하나 생성한다.
- 라벨에 들어갈 텍스트와 부모 위젯을 입력한다.

```
rbtn1.setChecked(True)
```

- `setChecked()`를 True로 설정하면 프로그램이 실행될 때 버튼이 선택되어 표시된다.

```
rbtn2.setText('Second Button')
```

- `setText()` 메서드를 통해서도 라벨의 텍스트를 설정할 수 있다.

## Section 04 위젯

### □QComboBox

- QComboBox는 작은 공간을 차지하면서, 여러 옵션들을 제공하고 그 중 하나의 옵션을 선택할 수 있도록 해주는 위젯이다. (QComboBox 공식 문서 참고)
- ex\_4\_5.py

```
1  import sys
2
3  from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QComboBox
4
5  class MyApp(QWidget):
6
7      def __init__(self):
8          super().__init__()
9          self.initUI()
10
11      def initUI(self):
12          self.lbl = QLabel('Option1', self)
13          self.lbl.move(50, 150)
```



# Section 04 위젯

## □QComboBox

- ex\_4\_5.py

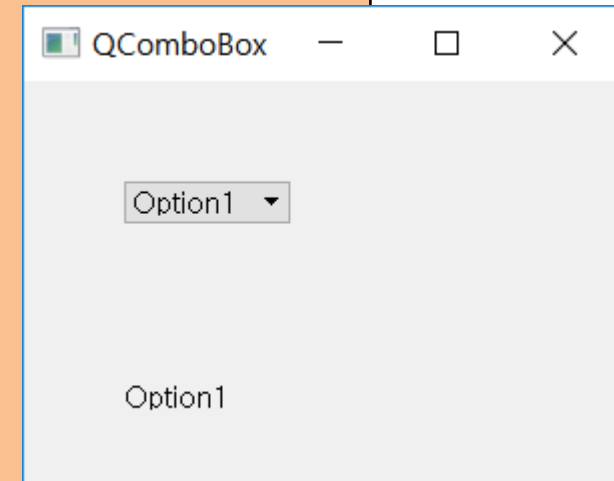
```
13         cb = QComboBox(self)
14         cb.addItem('Option1')
15         cb.addItem('Option2')
16         cb.addItem('Option3')
17         cb.addItem('Option4')
18         cb.move(50, 50)
19
20         cb.currentTextChanged.connect(self.onActivated)
21
22         self.setWindowTitle('QComboBox')
23         self.setGeometry(300, 300, 300, 200)
24         self.show()
25
```

## Section 04 위젯

### □QComboBox

- ex\_4\_5.py

```
26     def onActivated(self, text):
27         self.lbl.setText(text)
28         self.lbl.adjustSize()
29
30     if __name__ == '__main__':
31         app = QApplication(sys.argv)
32         ex = MyApp()
33         app.exec()
```



- 이 예제는 하나의 라벨과 콤보박스 위젯을 만들고, 콤보박스에서 선택한 항목이 라벨에 나타나도록 했다.

## Section 04 위젯

### □QComboBox

```
cb = QComboBox(self)
cb.addItem('Option1')
cb.addItem('Option2')
cb.addItem('Option3')
cb.addItem('Option4')
cb.move(50, 50)
```

- QComboBox 위젯을 하나 만들고, addItem() 메서드를 이용해서 선택 가능한 4개의 옵션들을 추가했다.

```
cb.currentTextChanged.connect(self.onActivated)
```

- 옵션을 선택하면, onActivated() 메서드가 호출된다.

## Section 04 위젯

---

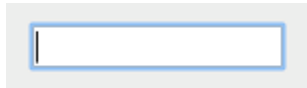
### □QComboBox

```
def onActivated(self, text):  
  
    self.lbl.setText(text)  
    self.lbl.adjustSize()
```

- **선택한 항목의 텍스트가 라벨에 나타나도록 하고, adjustSize() 메서드를 이용해서 라벨의 크기를 자동으로 조절하도록 한다.**

## Section 04 위젯

### □ QLineEdit



- QLineEdit은 한 줄의 문자열을 입력하고 수정할 수 있도록 하는 위젯이다. (QLineEdit 공식 문서 참고)
- echoMode()를 설정함으로써 '쓰기 전용' 영역으로 사용할 수 있다.
- 이 기능은 비밀번호와 같은 입력을 받을 때 유용하게 사용된다.
- setEchoMode() 메서드로 이러한 모드를 설정할 수 있으며, 입력값과 기능은 아래 표와 같다.

상수	값	설명
QLineEdit.Normal	0	입력된 문자를 표시합니다. (기본값)
QLineEdit.NoEcho	1	문자열을 표시하지 않습니다. 이 설정은 비밀번호의 글자수도 공개하지 않을 때 유용합니다.
QLineEdit.Password	2	입력된 문자 대신 비밀번호 가림용 문자를 표시합니다.
QLineEdit.PasswordEchoOnEdit	3	입력할 때만 문자를 표시하고, 수정 중에는 다른 문자를 표시합니다.

## Section 04 위젯

---

### □QLineEdit

- `maxLength()` 메서드로 입력되는 텍스트의 길이를 제한할 수 있고, `setValidator()` 메서드로 입력되는 텍스트의 종류를 제한할 수도 있다.
- `setText()` 또는 `insert()` 메서드로, 텍스트를 편집할 수 있고, `text()` 메서드로 입력된 텍스트를 가져올 수 있다.
- 만약 `echoMode`에 의해 입력되는 텍스트와 표시되는 텍스트가 다르다면, `displayText()` 메서드로 표시되는 텍스트를 가져올 수도 있다.
- `setSelection()`, `selectAll()` 메서드로 텍스트를 선택하거나, `cut()`, `copy()`, `paste()` 메서드를 통해 잘라내기, 복사하기, 붙여넣기 등의 동작을 수행할 수 있다.
- 또한 `setAlignment()` 메서드로 텍스트의 정렬을 설정할 수 있다.
- 텍스트가 변경되거나 커서가 움직일 때, `textChanged()`, `cursorPositionChanged()`와 같은 시그널이 발생한다.

## Section 04 위젯

### □QLineEdit

- 자주 사용되는 시그널

시그널	설명
cursorPositionChanged()	커서가 움직일 때 발생하는 신호를 발생합니다.
editingFinished()	편집이 끝났을 때 (Return/Enter 버튼이 눌릴 때) 신호를 발생합니다.
returnPressed()	Return/Enter 버튼이 눌릴 때 신호를 발생합니다.
selectionChanged()	선택 영역이 바뀔 때 신호를 발생합니다.
textChanged()	텍스트가 변경될 때 신호를 발생합니다.
textEdited()	텍스트가 편집될 때 신호를 발생합니다.

# Section 04 위젯

## □ QLineEdit

- ex\_4\_6.py

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QLineEdit
3
4  class MyApp(QWidget):
5      def __init__(self):
6          super().__init__()
7          self.initUI()
8
9      def initUI(self):
10         self.lbl = QLabel(self)
11         self.lbl.move(60, 40)
12
13         qle = QLineEdit(self)
14         qle.move(60, 100)
15         qle.textChanged[str].connect(self.onChanged)
```

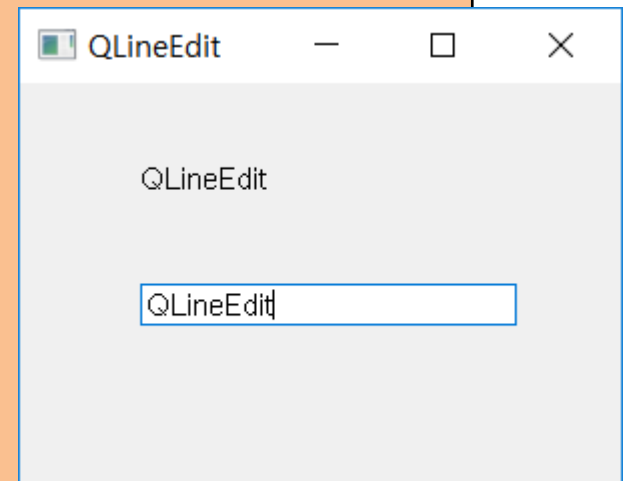


# Section 04 위젯

## □ QLineEdit

- ex\_4\_6.py

```
16
17     self.setWindowTitle('QLineEdit')
18     self.setGeometry(300, 300, 300, 200)
19     self.show()
20
21     def onChanged(self, text):
22         self.lbl.setText(text)
23         self.lbl.adjustSize()
24
25     if __name__ == '__main__':
26         app = QApplication(sys.argv)
27         ex = MyApp()
28         app.exec()
```



## Section 04 위젯

### □ QLineEdit

```
qle = QLineEdit(self)
```

- QLineEdit 위젯을 생성한다.

```
qle.textChanged[str].connect(self.onChanged)
```

- qle의 텍스트가 바뀌면, onChanged() 메서드를 호출한다.

```
def onChanged(self, text):
```

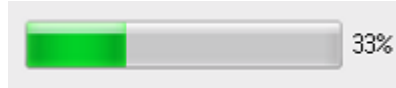
```
    self.lbl.setText(text)
```

```
    self.lbl.adjustSize()
```

- onChanged() 메서드 안에서, 입력된 'text'를 라벨 위젯(lbl)의 텍스트로 설정한다.
- 또한 adjustSize() 메서드로 텍스트의 길이에 따라 라벨의 길이를 조절한다.

## Section 04 위젯

### □QProgressBar



- 위의 그림은 각각 macOS와 Windows7의 기본적인 진행 표시줄(progress bar)을 나타낸다.
- 진행 표시줄은 시간이 걸리는 작업에 사용되는 위젯이다.
- QProgressBar 위젯은 수평, 수직의 진행 표시줄을 제공한다.
- setMinimum()과 setMaximum() 메서드로 진행 표시줄의 최소값과 최대값을 설정할 수 있으며, 또는 setRange() 메서드로 한 번에 범위를 설정할 수도 있다.
- 기본값은 0과 99이다.
- setValue() 메서드로 진행 표시줄의 진행 상태를 특정 값으로 설정할 수 있고, reset() 메서드는 초기 상태로 되돌린다.
- 진행 표시줄의 최소값과 최대값을 모두 0으로 설정하면, 진행 표시줄은 위의 그림과 같이 항상 진행 중인 상태로 표시된다.
- 이 기능은 다운로드하고 있는 파일의 용량을 알 수 없을 때 유용하게

# Section 04 위젯

## □QProgressBar

- ex\_4\_7.py

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget, QPushButton, QProgressBar
3  from PyQt6.QtCore import QBasicTimer
4
5  class MyApp(QWidget):
6      def __init__(self):
7          super().__init__()
8          self.initUI()
9
10     def initUI(self):
11         self.pbar = QProgressBar(self)
12         self.pbar.setGeometry(30, 40, 200, 25)
13
14         self.btn = QPushButton('Start', self)
15         self.btn.move(40, 80)
```

# Section 04 위젯

## □QProgressBar

- ex\_4\_7.py

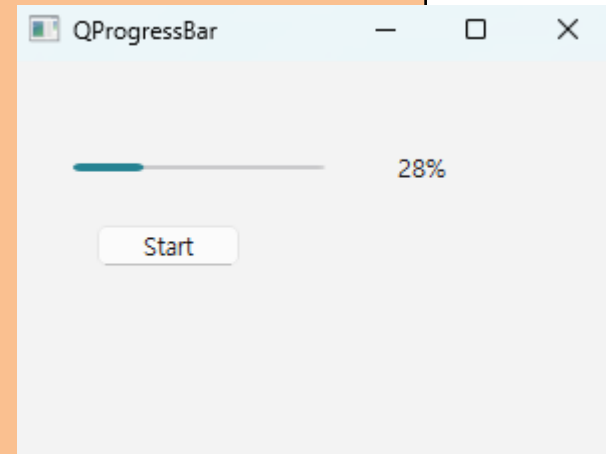
```
16         self.btn.clicked.connect(self.doAction)
17
18         self.timer = QBasicTimer()
19         self.step = 0
20
21         self.setWindowTitle('QProgressBar')
22         self.setGeometry(300, 300, 300, 200)
23         self.show()
24
25     def timerEvent(self, e):
26         if self.step >= 100:
27             self.timer.stop()
28             self.btn.setText('Finished')
29             return
30
```

# Section 04 위젯

## □QProgressBar

- ex\_4\_7.py

```
31         self.step = self.step + 1
32         self.pbar.setValue(self.step)
33
34     def doAction(self):
35         if self.timer.isActive():
36             self.timer.stop()
37             self.btn.setText('Start')
38         else:
39             self.timer.start(100, self)
40             self.btn.setText('Stop')
41
42     if __name__ == '__main__':
43         app = QApplication(sys.argv)
44         ex = MyApp()
45         app.exec()
```



## Section 04 위젯

---

### □QProgressBar

```
qle = QLineEdit(self)
```

- QProgressBar 생성자로 진행 표시줄을 하나 생성한다.

```
self.timer = QTimer()
```

- 진행 표시줄을 활성화하기 위해, 타이머 객체를 사용한다.

```
self.timer.start(100, self)
```

- 타이머 이벤트를 실행하기 위해, start() 메서드를 호출한다.
- 이 메서드는 두 개의 매개변수를 갖는데, 첫 번째는 종료시간이고 두 번째는 이벤트가 수행될 객체이다.

## Section 04 위젯

### □QProgressBar

```
def timerEvent(self, e):  
  
    if self.step >= 100:  
  
        self.timer.stop()  
        self.btn.setText('Finished')  
        return  
  
    self.step = self.step + 1  
    self.pbar.setValue(self.step)
```

- 각각의 QObject와 그 자손들은 timerEvent() 이벤트 핸들러를 갖는다.
- 타이머 이벤트에 반응하기 위해, 이벤트 핸들러를 재구성한다.



## Section 04 위젯

---

### □QProgressBar

```
def doAction(self):  
  
    if self.timer.isActive():  
        self.timer.stop()  
        self.btn.setText('Start')  
    else:  
        self.timer.start(100, self)  
        self.btn.setText('Stop')
```

- doAction() 메서드 안에서, 타이머를 시작하고 멈추도록 해준다.

## Section 04 위젯

---

### □QSlider & QDial

- QSlider는 수평 또는 수직 방향의 슬라이더를 제공한다.
- 슬라이더는 제한된 범위 안에서 값을 조절하는 위젯이다.



- 슬라이더의 틱(tick)의 간격을 조절하기 위해서는 `setTickInterval()` 메서드, 틱(tick)의 위치를 조절하기 위해서는 `setTickPosition()` 메서드를 사용한다.
- `setTickInterval()` 메서드의 입력값은 픽셀이 아니라 값을 의미한다.

## Section 04 위젯

### □QSlider & QDial

- setPosition() 메서드의 입력값과 기능

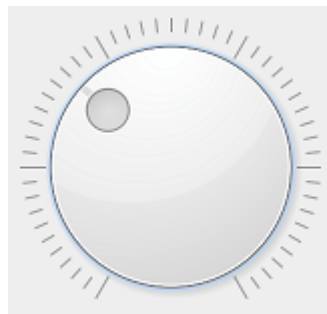
상수	값	설명
QSlider.NoTicks	0	틱을 표시하지 않습니다.
QSlider.TicksAbove	1	틱을 (수평) 슬라이더 위쪽에 표시합니다.
QSlider.TicksBelow	2	틱을 (수평) 슬라이더 아래쪽에 표시합니다.
QSlider.TicksBothSides	3	틱을 (수평) 슬라이더 양쪽에 표시합니다.
QSlider.TicksLeft	TicksAbove	틱을 (수직) 슬라이더 왼쪽에 표시합니다.
QSlider.TicksRight	TicksBelow	틱을 (수직) 슬라이더 오른쪽에 표시합니다.

## Section 04 위젯

---

### □QSlider & QDial

- QDial은 슬라이더를 둥근 형태로 표현한 다이얼 위젯이며, 기본적으로 같은 시그널과 슬롯, 메서드들을 공유한다.



- 위의 그림처럼 다이얼 위젯에 노치(notch)를 표시하기 위해서는 `setNotchesVisible()` 메서드를 사용한다.
- `True`로 설정하면 둥근 다이얼을 따라서 노치들이 표시된다.
- 기본적으로 노치는 표시되지 않도록 설정되어 있다.

## Section 04 위젯

---

### □QSlider & QDial

- 자주 쓰이는 시그널

시그널	설명
valueChanged()	슬라이더의 값이 변할 때 발생시킨다.
sliderPressed()	사용자가 슬라이더를 움직이기 시작할 때 발생시킨다.
sliderMoved()	사용자가 슬라이더를 움직이면 발생시킨다.
sliderReleased()	사용자가 슬라이더를 놓을 때 발생시킨다.

# Section 04 위젯

## □QSlider & QDial

- ex\_4\_8.py

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget, QSlider, QDial, QPushButton
3  from PyQt6.QtCore import Qt
4
5  class MyApp(QWidget):
6      def __init__(self):
7          super().__init__()
8          self.initUI()
9
10     def initUI(self):
11         self.slider = QSlider(Qt.Orientation.Horizontal, self)
12         self.slider.move(30, 30)
13         self.slider.setRange(0, 50)
14         self.slider.setSingleStep(2)
15
```

# Section 04 위젯

## □QSlider & QDial

- ex\_4\_8.py

```
16         self.dial = QDial(self)
17
18         self.dial.move(30, 50)
19
20         self.dial.setRange(0, 50)
21
22
23         btn = QPushButton('Default', self)
24         btn.move(35, 160)
25
26
27         self.slider.valueChanged.connect(self.dial.setValue)
28         self.dial.valueChanged.connect(self.slider.setValue)
29         btn.clicked.connect(self.button_clicked)
30
31
32         self.setWindowTitle('QSlider and QDial')
33         self.setGeometry(300, 300, 400, 200)
34         self.show()
```

## Section 04 위젯

### ❑ QSlider & QDial

- ex\_4\_8.py

```
31     def button_clicked(self):
32         self.slider.setValue(0)
33         self.dial.setValue(0)
34
35     if __name__ == '__main__':
36         app = QApplication(sys.argv)
37         ex = MyApp()
38         app.exec()
```

- 이 예제에서 슬라이더와 다이얼은 서로 연결되어 있어서 항상 같은 값을 가지고, 푸시 버튼을 누르면 두 위젯의 값이 모두 0이 된다.



## Section 04 위젯

### □QSlider & QDial

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

- QSlider 위젯을 하나 생성한다.
- Qt.Horizontal 또는 Qt.Vertical을 입력해줌으로써 수평 또는 수직 방향을 설정한다.

```
self.slider.setRange(0, 50)  
self.slider.setSingleStep(2)
```

- setRange() 메서드로 값의 범위를 0에서 50까지로 설정한다.
- setSingleStep() 메서드는 조절 가능한 최소 단위를 설정한다.

```
self.dial = QDial(self)
```

- QDial 위젯을 하나 생성한다.

## Section 04 위젯

### □QSlider & QDial

```
self.dial.setRange(0, 50)
```

- 슬라이더와 마찬가지로 `setRange()` 메서드로 범위를 설정한다.

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

- 슬라이더와 다이얼의 값이 변할 때 발생하는 시그널을 각각 다이얼과 슬라이더의 값을 조절해주는 메서드 (`setValue`)에 서로 연결함으로써 두 위젯의 값이 언제나 일치하도록 한다.

```
btn.clicked.connect(self.button_clicked)
```

- 'Default' 푸시 버튼을 클릭하면 발생하는 시그널을 `button_clicked` 메서드에 연결한다.

## Section 04 위젯

---

### □QSlider & QDial

```
def button_clicked(self):  
    self.slider.setValue(0)  
    self.dial.setValue(0)
```

- button\_clicked() 메서드는 슬라이더와 다이얼의 값을 모두 0으로 조절한다.
- 따라서 'Default' 푸시 버튼을 클릭하면, 슬라이더와 다이얼의 값이 0으로 초기화된다.

## Section 04 위젯

---

### □QSplitter

- QSplitter 클래스는 스플리터 위젯을 구현한다.
- 스플리터 (splitter)는 경계를 드래그해서 자식 위젯의 크기를 조절할 수 있도록 한다.
- 예제를 통해 위젯을 네 개의 작은 영역으로 쪼개어 보자.

# Section 04 위젯

## □QSplitter

- ex\_4\_9.py

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget, QHBoxLayout, QFrame, QSplitter
3  from PyQt6.QtCore import Qt
4
5  class MyApp(QWidget):
6      def __init__(self):
7          super().__init__()
8          self.initUI()
9
10     def initUI(self):
11         hbox = QHBoxLayout()
12
13         top = QFrame()
14         top.setFrameShape(QFrame.Shape.Box)
15
```

# Section 04 위젯

## □QSplitter

- ex\_4\_9.py

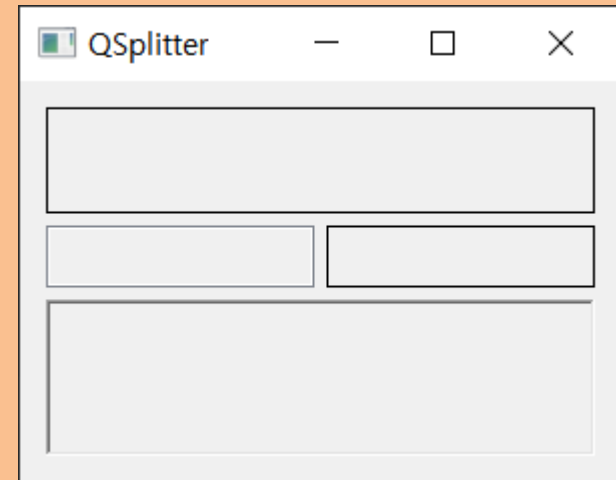
```
16         midleft = QFrame()
17         midleft.setFrameShape(QFrame.Shape.StyledPanel)
18
19         midright = QFrame()
20         midright.setFrameShape(QFrame.Shape.Panel)
21
22         bottom = QFrame()
23         bottom.setFrameShape(QFrame.Shape.WinPanel)
24         bottom.setFrameShadow(QFrame.Shadow.Sunken)
25
26         splitter1 = QSplitter(Qt.Orientation.Horizontal)
27         splitter1.addWidget(midleft)
28         splitter1.addWidget(midright)
29
30         splitter2 = QSplitter(Qt.Orientation.Vertical)
```

# Section 04 위젯

## □QSplitter

- ex\_4\_9.py

```
31         splitter2.addWidget(top)
32         splitter2.addWidget(splitter1)
33         splitter2.addWidget(bottom)
34
35         hbox.addWidget(splitter2)
36         self.setLayout(hbox)
37
38         self.setGeometry(300, 300, 300, 200)
39         self.setWindowTitle('QSplitter')
40         self.show()
41
42     if __name__ == '__main__':
43         app = QApplication(sys.argv)
44         ex = MyApp()
45         app.exec()
```



## Section 04 위젯

### □QSplitter

```
top = QFrame()  
top.setFrameShape(QFrame.Shape.Box)  
  
midleft = QFrame()  
midleft.setFrameShape(QFrame.Shape.StyledPanel)  
  
midright = QFrame()  
midright.setFrameShape(QFrame.Shape.Panel)  
  
bottom = QFrame()  
bottom.setFrameShape(QFrame.Shape.WinPanel)  
bottom.setFrameShadow(QFrame.Shadow.Sunken)
```

- 각 영역에 들어갈 프레임을 생성한다.
- 프레임의 형태와 그림자의 스타일을 setFrameShape과 setFrameShadow를 이용해서 설정할 수 있다.



# Section 04 위젯

## □QSplitter

- 프레임의 형태와 그림자

0				1				2				3				lineWidth()
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	midLineWidth()
																Box, Plain
																Box, Raised
																Box, Sunken
																Panel, Plain
																Panel, Raised
																Panel, Sunken
																WinPanel, Plain
																WinPanel, Raised
																WinPanel, Sunken
																HLine, Plain
																HLine, Raised
																HLine, Sunken
																VLine, Plain
																VLine, Raised
																VLine, Sunken
																StyledPanel, Plain
																StyledPanel, Raised
																StyledPanel, Sunken

## Section 04 위젯

### □QSplitter

```
splitter1 = QSplitter(Qt.Orientation.Horizontal)
splitter1.addWidget(midleft)
splitter1.addWidget(midright)

splitter2 = QSplitter(Qt.Orientation.Vertical)
splitter2.addWidget(top)
splitter2.addWidget(splitter1)
splitter2.addWidget(bottom)
```

- QSplitter를 이용해서 수평 방향으로 쪼개고, 왼쪽에는 midleft, 오른쪽에는 midright 프레임 위젯을 넣어준다.
- 다음, 수직 방향으로 쪼개고, top, splitter1, bottom 3개의 프레임 위젯을 넣어준다.

## Section 04 위젯

---

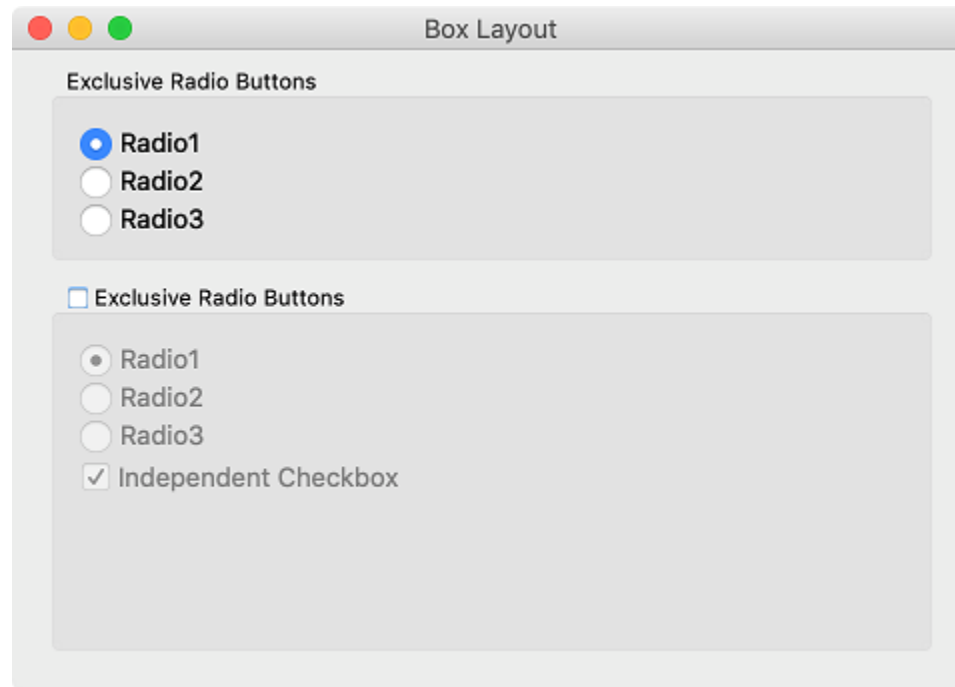
### □QGroupBox

- 그룹 박스는 상단 타이틀과 단축키 (shortcut)를 제공하며, 그 안에 다양한 위젯들을 나타낼 수 있다. (QGroupBox 공식 문서 참고)
- QGroupBox 클래스는 제목과 제목의 정렬을 설정한다.
- 그룹 박스는 체크 가능하도록 설정할 수 있는데, 그룹 박스의 체크 여부에 따라 그 그룹 박스 안에 있는 위젯들이 사용 가능하거나 불가능해진다.
- 공간을 절약하기 위해 flat 속성을 활성화할 수 있는데, 보통 프레임의 왼쪽, 오른쪽, 아래쪽 가장자리가 없게 표현한다.

## Section 04 위젯

### □QGroupBox

- 그림과 같이 QGroupBox 위젯은 제목을 갖는 그룹 박스 프레임を提供한다.



# Section 04 위젯

## □QGroupBox

- ex\_4\_10.py

```
1      import sys
2
3      from PyQt6.QtWidgets import (QApplication, QWidget, QGroupBox, QRadioButton
4      , QCheckBox, QPushButton, QMenu, QGridLayout, QVBoxLayout)
5
6      class MyApp(QWidget):
7
8          def __init__(self):
9              super().__init__()
10             self.initUI()
11
12         def initUI(self):
13             grid = QGridLayout()
14             grid.addWidget(self.createFirstExclusiveGroup(), 0, 0)
15             grid.addWidget(self.createSecondExclusiveGroup(), 1, 0)
16             grid.addWidget(self.createNonExclusiveGroup(), 0, 1)
17             grid.addWidget(self.createPushButtonGroup(), 1, 1)
```

# Section 04 위젯

## □QGroupBox

- ex\_4\_10.py

```
16         self.setLayout(grid)
17
18         self.setWindowTitle('Box Layout')
19         self.setGeometry(300, 300, 480, 320)
20         self.show()
21
22     def createFirstExclusiveGroup(self):
23         groupbox = QGroupBox('Exclusive Radio Buttons')
24         radio1 = QRadioButton('Radio1')
25         radio2 = QRadioButton('Radio2')
26         radio3 = QRadioButton('Radio3')
27         radio1.setChecked(True)
28
29         vbox = QVBoxLayout()
30         vbox.addWidget(radio1)
```

# Section 04 위젯

## □QGroupBox

- ex\_4\_10.py

```
31         vbox.addWidget(radio2)
32         vbox.addWidget(radio3)
33         groupbox.setLayout(vbox)
34
35         return groupbox
36
37     def createSecondExclusiveGroup(self):
38         groupbox = QGroupBox('Exclusive Radio Buttons')
39         groupbox.setCheckable(True)
40         groupbox.setChecked(False)
41
42         radio1 = QRadioButton('Radio1')
43         radio2 = QRadioButton('Radio2')
44         radio3 = QRadioButton('Radio3')
45         radio1.setChecked(True)
```

# Section 04 위젯

## □QGroupBox

- ex\_4\_10.py

```
46         checkbox = QCheckBox('Independent Checkbox')
47         checkbox.setChecked(True)
48
49         vbox = QVBoxLayout()
50         vbox.addWidget(radio1)
51         vbox.addWidget(radio2)
52         vbox.addWidget(radio3)
53         vbox.addWidget(checkbox)
54         vbox.addStretch(1)
55         groupbox.setLayout(vbox)
56
57         return groupbox
58
59     def createNonExclusiveGroup(self):
60         groupbox = QGroupBox('Non-Exclusive Checkboxes')
```



# Section 04 위젯

## □QGroupBox

- ex\_4\_10.py

```
61         groupbox.setFlat(True)
62
63         checkbox1 = QCheckBox('Checkbox1')
64         checkbox2 = QCheckBox('Checkbox2')
65         checkbox2.setChecked(True)
66         tristatebox = QCheckBox("Tri-state Button")
67         tristatebox.setTristate(True)
68
69         vbox = QVBoxLayout()
70         vbox.addWidget(checkbox1)
71         vbox.addWidget(checkbox2)
72         vbox.addWidget(tristatebox)
73         vbox.addStretch(1)
74         groupbox.setLayout(vbox)
75
```

# Section 04 위젯

## □QGroupBox

- ex\_4\_10.py

```
76         return groupbox
77
78     def createPushButtonGroup(self):
79         groupbox = QGroupBox('Push Buttons')
80         groupbox.setCheckable(True)
81         groupbox.setChecked(True)
82
83         pushbutton = QPushButton('Normal Button')
84         togglebutton = QPushButton('Toggle Button')
85         togglebutton.setCheckable(True)
86         togglebutton.setChecked(True)
87         flatbutton = QPushButton('Flat Button')
88         flatbutton.setFlat(True)
89         popupbutton = QPushButton('Popup Button')
90         menu = QMenu(self)
```

# Section 04 위젯

## □QGroupBox

- ex\_4\_10.py

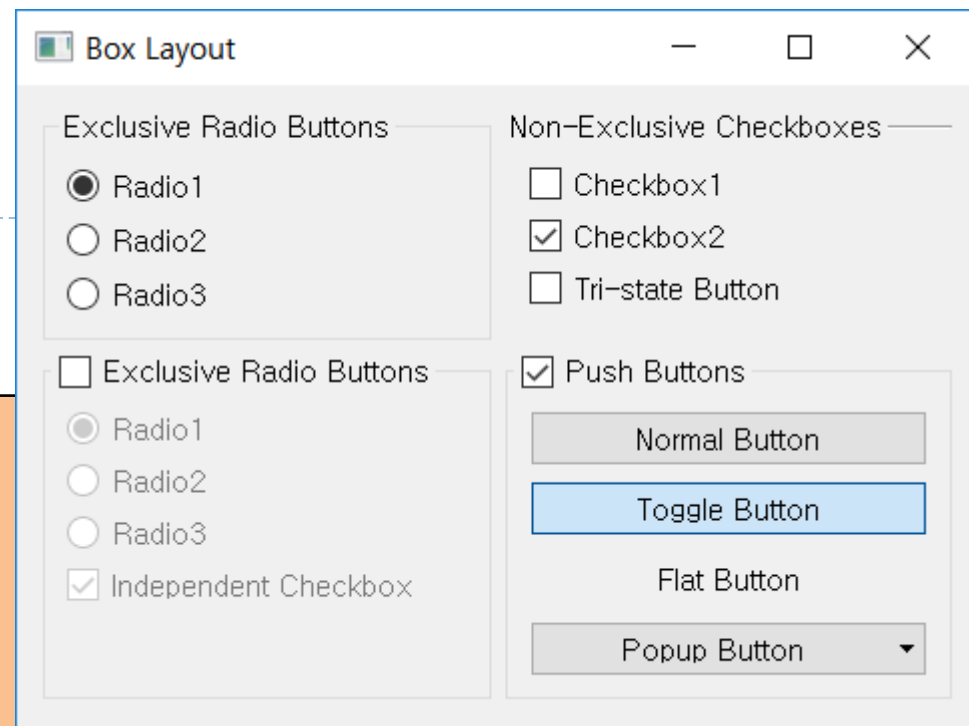
```
91         menu.addAction('First Item')
92         menu.addAction('Second Item')
93         menu.addAction('Third Item')
94         menu.addAction('Fourth Item')
95         popupbutton.setMenu(menu)
96
97         vbox = QVBoxLayout()
98         vbox.addWidget(pushbutton)
99         vbox.addWidget(togglebutton)
100        vbox.addWidget(flatbutton)
101        vbox.addWidget(popupbutton)
102        vbox.addStretch(1)
103        groupbox.setLayout(vbox)
104
105    return groupbox
```

## Section 04 위젯

### ❑ QGroupBox

- ex\_4\_10.py

```
106
107     if __name__ == '__main__':
108         app = QApplication(sys.argv)
109         ex = MyApp()
110         app.exec()
```



- QGroupBox 클래스를 이용해서 버튼의 종류에 따라 네 개의 그룹 박스를 만들었다.

## Section 04 위젯

---

### □QGroupBox

```
grid = QGridLayout()
grid.addWidget(self.createFirstExclusiveGroup(), 0, 0)
grid.addWidget(self.createSecondExclusiveGroup(), 1, 0)
grid.addWidget(self.createNonExclusiveGroup(), 0, 1)
grid.addWidget(self.createPushButtonGroup(), 1, 1)

self.setLayout(grid)
```

- **그리드 레이아웃을 이용해서 그룹박스를 배치한다.**
- **각 메서드를 통해 만들어지는 그룹박스가 addWidget()을 통해 각 위치로 배치된다.**

## Section 04 위젯

### □QGroupBox

```
def createFirstExclusiveGroup(self):  
    groupbox = QGroupBox('Exclusive Radio Buttons')  
  
    radio1 = QRadioButton('Radio1')  
    radio2 = QRadioButton('Radio2')  
    radio3 = QRadioButton('Radio3')  
    radio1.setChecked(True)  
  
    vbox = QVBoxLayout()  
    vbox.addWidget(radio1)  
    vbox.addWidget(radio2)  
    vbox.addWidget(radio3)  
    groupbox.setLayout(vbox)  
  
    return groupbox
```

- createFirstExclusiveGroup() 메서드는 배타적인 라디오버튼을 갖는 그룹박스를 생성한다.

▶ 158 • 메서드는 먼저 그룹박스 (groupbox)를 하나 만들고, 버튼을 만든 다음 수직 박스, 개인식용을 통해 배치합니다.

## Section 04 위젯

---

### □QGroupBox

- 메서드는 먼저 그룹박스 (groupbox)를 하나 만들고, 버튼을 만든 다음 수직 박스 레이아웃을 통해 배치한다.
- 마지막으로 수직 박스 레이아웃 (vbox)를 그룹 박스의 레이아웃으로 설정한다.
- 세 개의 라디오버튼을 만들고 수직으로 배치했다.

## Section 04 위젯

### □QGroupBox

```
def createSecondExclusiveGroup(self):  
    groupbox = QGroupBox('Exclusive Radio Buttons')  
    groupbox.setCheckable(True)  
    groupbox.setChecked(False)
```

- createSecondExclusiveGroup() 메서드는 세 개의 라디오버튼과 한 개의 체크박스를 갖는 그룹박스를 생성한다.
- setCheckable() 메서드를 이용해서 groupbox도 선택 가능하도록 할 수 있다.

```
def createNonExclusiveGroup(self):  
    groupbox = QGroupBox('Non-Exclusive Checkboxes')  
    groupbox.setFlat(True)
```

- createNonExclusiveGroup() 메서드는 배타적이지 않은 체크박스들을 갖는 그룹박스를 생성한다.

▶ 160 setFlat()을 이용해서 그룹박스를 평평하게 보이도록 한다.



## Section 04 위젯

---

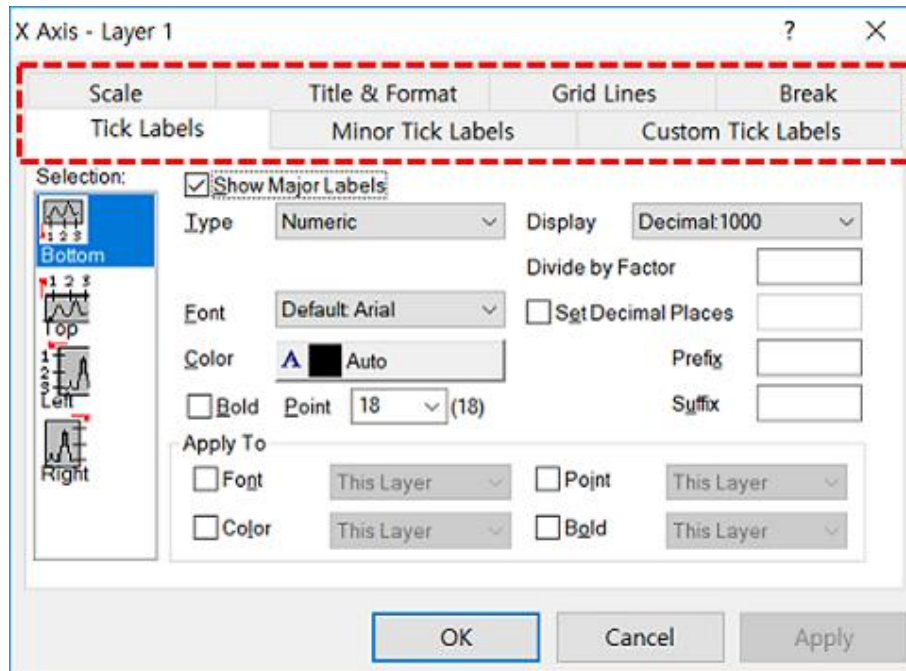
### □QGroupBox

```
def createPushButtonGroup(self):  
    groupbox = QGroupBox('Push Buttons')  
    groupbox.setCheckable(True)  
    groupbox.setChecked(True)
```

- createPushButtonGroup() 메서드는 여러 개의 푸시버튼을 갖는 그룹 박스를 생성한다.
- setCheckable()과 setChecked()를 이용해서, 그룹 박스를 선택 가능하게, 그리고 실행했을 때 선택되어 있도록 설정한다.

## Section 04 위젯

### □QTabWidget



- GUI 프로그램을 사용하다보면 위의 그림과 같이 탭(Tab)이 있는 창을 볼 수 있다.
- 이러한 탭은 프로그램 안의 구성요소들이 많은 면적을 차지하지 않으면서, 그것들을 카테고리에 따라 분류할 수 있기 때문에 유용하게 사용될 수 있다.
- 간단한 예제를 통해 두 개의 탭을 갖는 위젯을 하나 만들어보자.

# Section 04 위젯

## □QTabWidget

- ex\_4\_11.py

```
1      import sys
2      from PyQt6.QtWidgets import QApplication, QWidget, QTabWidget, QVBoxLayout
3
4      class MyApp(QWidget):
5          def __init__(self):
6              super().__init__()
7              self.initUI()
8
9          def initUI(self):
10             tab1 = QWidget()
11             tab2 = QWidget()
12
13             tabs = QTabWidget()
14             tabs.addTab(tab1, 'Tab1')
15             tabs.addTab(tab2, 'Tab2')
```

# Section 04 위젯

## □QTabWidget

- ex\_4\_11.py

```
16
17     vbox = QVBoxLayout()
18     vbox.addWidget(tabs)
19
20     self.setLayout(vbox)
21
22     self.setWindowTitle('QTabWidget')
23     self.setGeometry(300, 300, 300, 200)
24     self.show()
25
26 if __name__ == '__main__':
27     app = QApplication(sys.argv)
28     ex = MyApp()
29     app.exec()
```

## Section 04 위젯

---

### □QTabWidget

```
tab1 = QWidget()  
tab2 = QWidget()
```

- 각 탭에 위치할 두 개의 위젯을 생성한다.

```
tabs = QTabWidget()  
tabs.addTab(tab1, 'Tab1')  
tabs.addTab(tab2, 'Tab2')
```

- QTabWidget()을 이용해서 탭을 만들어주고, addTab()을 이용해서 'Tab1'과 'Tab2'를 tabs에 추가해준다.

## Section 04 위젯

---

### □QTabWidget

```
vbox = QVBoxLayout()  
vbox.addWidget(tabs)  
  
self.setLayout(vbox)
```

- 수직 박스 레이아웃을 하나 만들어서 탭 위젯 (tabs)을 넣어준다.
- 그리고 수직 박스(vbox)를 위젯의 레이아웃으로 설정한다.

## Section 04 위젯

### □ QPixmap

- QPixmap은 이미지를 다룰 때 사용되는 위젯이다.
- 지원하는 파일 형식은 아래와 같다.
- 어떤 이미지 형식은 '읽기'만 가능하다.

Format	Description	Qt's support
BMP	Windows Bitmap	Read/write
GIF	Graphic Interchange Format (optional)	Read
JPG	Joint Photographic Experts Group	Read/write
JPEG	Joint Photographic Experts Group	Read/write
PNG	Portable Network Graphics	Read/write
PBM	Portable Bitmap	Read
PGM	Portable Graymap	Read
PPM	Portable Pixmap	Read/write
XBM	X11 Bitmap	Read/write
XPM	X11 Pixmap	Read/write

# Section 04 위젯

## □ QPixmap

- ex\_4\_12.py

```
1      import sys
2      from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QVBoxLayout
3      from PyQt6.QtGui import QPixmap
4      from PyQt6.QtCore import Qt
5
6      class MyApp(QWidget):
7          def __init__(self):
8              super().__init__()
9              self.initUI()
10
11         def initUI(self):
12             pixmap = QPixmap('./images/model3.jpg')
13
14             lbl_img = QLabel()
15             lbl_img.setPixmap(pixmap)
```



# Section 04 위젯

## □ QPixmap

```
• 16         lbl_size = QLabel('Width: '+str(pixmap.width())+', Height: '+str(pixmap.height()))
17         lbl_size.setAlignment(Qt.AlignmentFlag.AlignVCenter)
18
19         vbox = QVBoxLayout()
20         vbox.addWidget(lbl_img)
21         vbox.addWidget(lbl_size)
22         self.setLayout(vbox)
23
24         self.setWindowTitle('QPixmap')
25         self.move(300, 300)
26         self.show()
27
28     if __name__ == '__main__':
29         app = QApplication(sys.argv)
30         ex = MyApp()
31         app.exec()
```

## Section 04 위젯

---

### □ QPixmap

```
pixmap = QPixmap('./images/model3.jpg')
```

- 파일 이름을 입력해주고, QPixmap 객체 (pixmap)를 하나 생성한다.

```
lbl_img = QLabel()  
lbl_img.setPixmap(pixmap)
```

- 라벨을 하나 만들고, setPixmap을 이용해서 pixmap을 라벨에 표시될 이미지로 설정한다.

## Section 04 위젯

### □ QPixmap

```
lbl_size = QLabel('Width: '+str(pixmap.width())+', Height: '+str(pixmap.height()))  
lbl_size.setAlignment(Qt.AlignCenter)
```

- width()와 height()는 이미지의 너비, 높이를 반환한다.
- 너비, 높이를 표시하는 라벨 (lbl\_size)를 하나 만들고, setAlignment 메서드를 이용해서 가운데 정렬로 설정한다.

```
vbox = QVBoxLayout()  
vbox.addWidget(lbl)  
self.setLayout(vbox)
```

- 수평 박스 레이아웃을 하나 만들고 라벨을 배치한다.
- setLayout()을 이용해서 수평 박스(hbox)를 창의 레이아웃으로 지정한다.

## Section 04 위젯

### □QCalendarWidget

- QCalendarWidget을 이용해서 사용자가 날짜를 선택할 수 있도록 달력을 표시할 수 있다.
- 달력은 월 단위로 표시되고, 처음 실행될 때 현재의 연도, 월, 날짜로 선택되어 있다.

	November 2012						
	Mon	Tue	Wed	Thu	Fri	Sat	Sun
44	29	30	31	1	2	3	4
45	5	6	7	8	9	10	11
46	12	13	14	15	16	17	18
47	19	20	21	22	23	24	25
48	26	27	28	29	30	1	2
49	3	4	5	6	7	8	9

# Section 04 위젯

## □QCalendarWidget

- ex\_4\_13.py

```
1      import sys
2      from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QVBoxLayout, QCalendar
3      from PyQt6.QtCore import Qdate
4
5      class MyApp(QWidget):
6          def __init__(self):
7              super().__init__()
8              self.initUI()
9
10         def initUI(self):
11             cal = QCalendarWidget(self)
12             cal.setGridVisible(True)
13             cal.clicked[QDate].connect(self.showDate)
14
```

## Section 04 위젯

### □QCalendarWidget

- ex\_4\_13.py

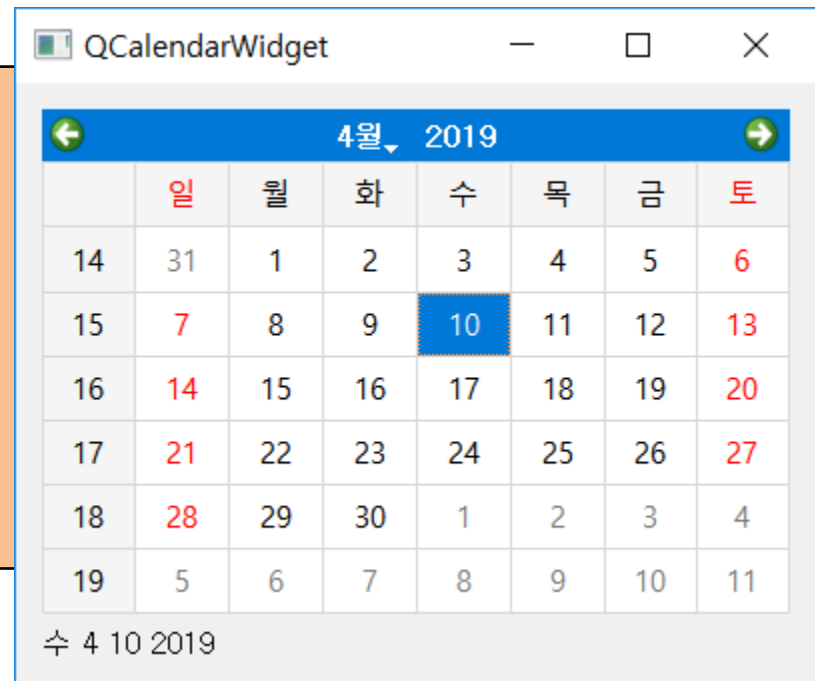
```
15         self.lbl = QLabel(self)
16
17         date = cal.selectedDate()
18
19         self.lbl.setText(date.toString())
20
21         vbox = QVBoxLayout()
22         vbox.addWidget(cal)
23         vbox.addWidget(self.lbl)
24
25         self.setLayout(vbox)
26
27         self.setWindowTitle('QCalendarWidget')
28         self.setGeometry(300, 300, 400, 300)
29         self.show()
```

# Section 04 위젯

## □QCalendarWidget

- ex\_4\_13.py

```
29     def showDate(self, date):
30         self.lbl.setText(date.toString())
31
32     if __name__ == '__main__':
33         app = QApplication(sys.argv)
34         ex = MyApp()
35         app.exec()
```



## Section 04 위젯

### □QCalendarWidget

```
cal = QCalendarWidget(self)
cal.setGridVisible(True)
cal.clicked[QDate].connect(self.showDate)
```

- QCalenderWidget의 객체(cal)를 하나 생성한다.
- setGridVisible(True)로 설정하면, 날짜 사이에 그리드가 표시된다.
- 날짜를 클릭했을 때 showDate 메서드가 호출되도록 연결한다.

```
self.lbl = QLabel(self)
date = cal.selectedDate()
self.lbl.setText(date.toString())
```

- selectedDate는 현재 선택된 날짜 정보를 갖고 있다. (디폴트는 현재 날짜)
- 현재 날짜 정보를 라벨에 표시되도록 해준다.



## Section 04 위젯

---

### □QCalendarWidget

```
vbox = QVBoxLayout()  
vbox.addWidget(cal)  
vbox.addWidget(self.lbl)
```

- 수직 박스 레이아웃을 이용해서, 달력과 라벨을 수직으로 배치해준다.

```
def showDate(self, date):  
  
    self.lbl.setText(date.toString())
```

- showDate 메서드에서, 날짜를 클릭할 때마다 라벨 텍스트가 선택한 날짜(date.toString())로 표시되도록 한다.

# Section 04 위젯

## □QSpinBox

- QSpinBox 클래스는 정수를 선택, 조절하도록 하는 스펀 박스 위젯을 제공한다.
- ex\_4\_14.py

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QSpinBox, QVBoxLayout
3
4  class MyApp(QWidget):
5      def __init__(self):
6          super().__init__()
7          self.initUI()
8
9      def initUI(self):
10         self.lbl1 = QLabel('QSpinBox')
11         self.spinbox = QSpinBox()
12         self.spinbox.setMinimum(-10)
13         self.spinbox.setMaximum(30)
```

# Section 04 위젯

## □QSpinBox

- QSpinBox 클래스는 정수를 선택, 조절하도록 하는 스펀 박스 위젯을 제공한다.
- ex\_4\_14.py

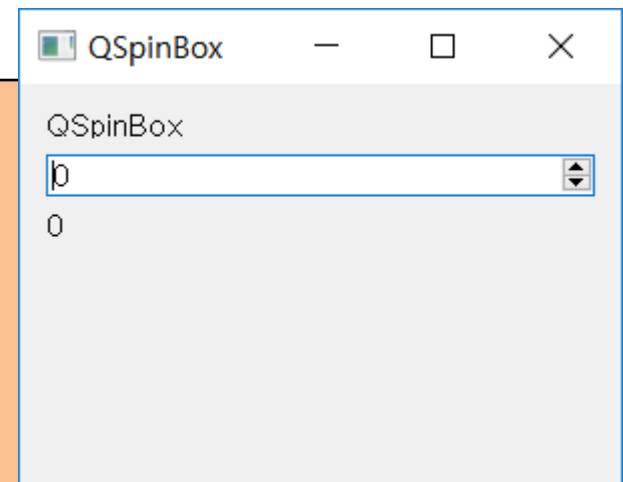
```
14         self.spinbox.setSingleStep(2)
15         self.lbl2 = QLabel('0')
16
17         self.spinbox.valueChanged.connect(self.value_changed)
18
19         vbox = QVBoxLayout()
20         vbox.addWidget(self.lbl1)
21         vbox.addWidget(self.spinbox)
22         vbox.addWidget(self.lbl2)
23         vbox.addStretch()
24
25         self.setLayout(vbox)
26
```

## Section 04 위젯

### □QSpinBox

- QSpinBox 클래스는 정수를 선택, 조절하도록 하는 스프인 박스 위젯을 제공한다.
- ex\_4\_14.py

```
27         self.setWindowTitle('QSpinBox')
28         self.setGeometry(300, 300, 300, 200)
29         self.show()
30
31         def value_changed(self):
32             self.lbl2.setText(str(self.spinbox.value()))
33
34     if __name__ == '__main__':
35         app = QApplication(sys.argv)
36         ex = MyApp()
37         app.exec()
```



## Section 04 위젯

### □QSpinBox

```
self.spinbox = QSpinBox()  
self.spinbox.setMinimum(-10)  
self.spinbox.setMaximum(30)
```

- QSpinBox 객체 (self.spinbox)를 하나 생성한다.
- setMinimum()과 setMaximum() 메서드를 이용해서 선택 범위를 제한할 수 있다.
- 최소값은 0, 최대값은 99가 디폴트이다.

```
# self.spinbox.setRange(-10, 30)
```

- setRange() 메서드는 setMinimum()과 setMaximum()을 합쳐놓은 것과 같다.

## Section 04 위젯

---

### □QSpinBox

```
self.spinbox.setSingleStep(2)
```

- **setSingleStep()**을 이용해서 한 스텝을 2로 설정한다.
- 스펀 박스의 경우, 한 스텝으로 설정할 수 있는 최소값은 1이다.

```
self.spinbox.valueChanged.connect(self.value_changed)
```

- 스펀 박스 위젯의 값이 변경될 때 발생하는 시그널 (valueChanged)을 self.value\_changed 메서드에 연결한다.

## Section 04 위젯

---

### □QSpinBox

```
vbox = QVBoxLayout()  
vbox.addWidget(self.lbl1)  
vbox.addWidget(self.spinbox)  
vbox.addWidget(self.lbl2)  
vbox.addStretch()  
  
self.setLayout(vbox)
```

- 수직 박스 레이아웃을 이용해서 라벨 두 개와 스펀 박스 위젯을 수직으로 배치하고, 전체 위젯의 레이아웃으로 설정한다.

## Section 04 위젯

---

### □QSpinBox

```
def value_changed(self):  
    self.lbl2.setText(str(self.spinbox.value()))
```

- 스펀 박스의 값이 변경될 때, self.lbl2의 텍스트를 스펀 박스의 값 [self.spinbox.value()]으로 설정하도록 한다.



## Section 04 위젯

### □QDoubleSpinBox

- QDoubleSpinBox 클래스는 실수를 선택, 조절하도록 하는 더블 스펀 박스 위젯을 제공한다.
- ex\_4\_15.py

```
1      import sys
2
3      from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QDoubleSpinBox, QVBoxLayout
4
5      class MyApp(QWidget):
6
7          def __init__(self):
8              super().__init__()
9              self.initUI()
10
11          def initUI(self):
12              self.lbl1 = QLabel('QDoubleSpinBox')
13              self.dspinbox = QDoubleSpinBox()
14              self.dspinbox.setRange(0, 100)
```

# Section 04 위젯

## □QDoubleSpinBox

- ex\_4\_15.py

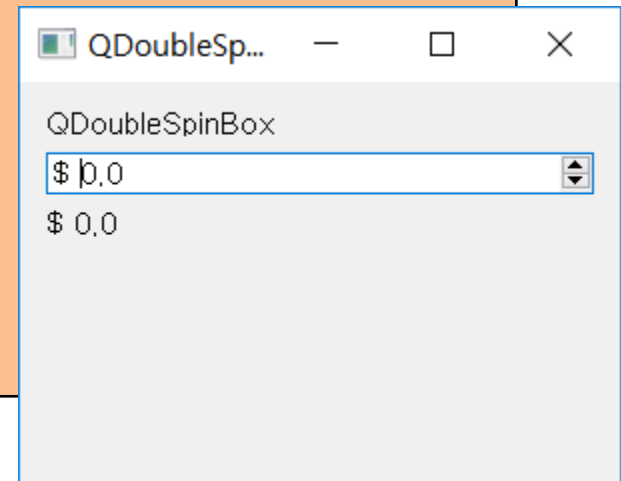
```
13         self.dspinbox.setSingleStep(0.5)
14
15         self.dspinbox.setPrefix('$ ')
16
17         self.dspinbox.setDecimals(1)
18         self.lbl2 = QLabel('$ 0.0')
19
20         self.dspinbox.valueChanged.connect(self.value_changed)
21
22         vbox = QVBoxLayout()
23         vbox.addWidget(self.lbl1)
24         vbox.addWidget(self.dspinbox)
25         vbox.addWidget(self.lbl2)
26         vbox.addStretch()
27
28         self.setLayout(vbox)
```

## Section 04 위젯

### □QDoubleSpinBox

- ex\_4\_15.py

```
28         self.setWindowTitle('QDoubleSpinBox')
29         self.setGeometry(300, 300, 300, 200)
30         self.show()
31
32         def value_changed(self):
33             self.lbl2.setText('$ ' + str(self.dspinbox.value()))
34
35     if __name__ == '__main__':
36         app = QApplication(sys.argv)
37         ex = MyApp()
38         sys.exit(app.exec())
```



## Section 04 위젯

### □QDoubleSpinBox

```
self.dspinbox = QDoubleSpinBox()  
self.dspinbox.setRange(0, 100)  
self.dspinbox.setSingleStep(0.5)
```

- QDoubleSpinBox 객체 (self.dspinbox)를 하나 생성한다.
- setRange() 메서드를 이용해서 선택 범위를 제한할 수 있다.
- 최소값은 0.0, 최대값은 99.99가 디폴트이다.
- setSingleStep() 메서드를 이용해서 한 스텝을 0.5로 설정한다.

```
self.dspinbox.setPrefix('$ ')  
self.dspinbox.setDecimals(1)
```

- setPrefix()를 이용해서, 숫자 앞에 올 문자를 설정할 수 있다.
- setSuffix()는 숫자 뒤에 문자가 오도록 한다.
- setDecimals()를 이용해서 소수점 아래 표시될 자리수를 정해준다.

## Section 04 위젯

### □QDoubleSpinBox

```
self.dspinbox.valueChanged.connect(self.value_changed)
```

- 더블 스펀 박스 위젯의 값이 변경될 때 발생하는 시그널 (valueChanged)을 self.value\_changed 메서드에 연결한다.

```
vbox = QVBoxLayout()  
vbox.addWidget(self.lbl1)  
vbox.addWidget(self.dspinbox)  
vbox.addWidget(self.lbl2)  
vbox.addStretch()  
  
self.setLayout(vbox)
```

- 수직 박스 레이아웃을 이용해서 라벨 두 개와 스펀 박스 위젯을 수직으로 배치하고, 전체 위젯의 레이아웃으로 설정한다.

## Section 04 위젯

### □QDateEdit

- QDateEdit 위젯은 사용자에게 날짜를 선택, 편집하도록 할 때 사용한다.
- 예제에서 QDateEdit 객체를 하나 만들고, 현재 날짜로 표시되도록 설정해보자.
- ex4\_16.py

```
1      import sys
2
3      from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QDateEdit, QVBoxLayout
4
5      from PyQt6.QtCore import QDate
6
7      class MyApp(QWidget):
8
9          def __init__(self):
10             super().__init__()
11             self.initUI()
12
13             def initUI(self):
14                 lbl = QLabel('QDateEdit')
```

# Section 04 위젯

## □QDateEdit

- ex4\_16.py

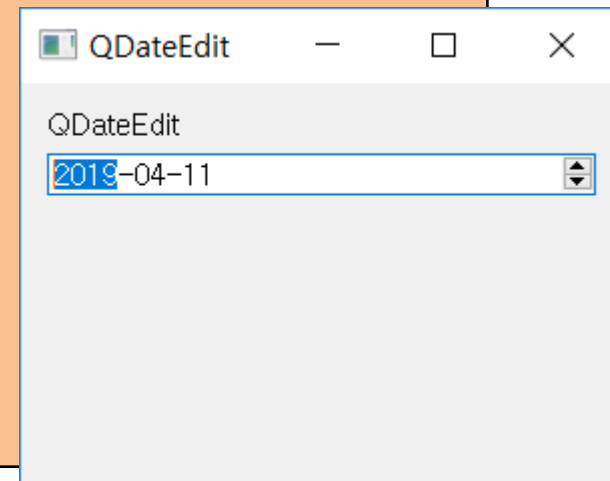
```
12
13     dateedit = QDateEdit(self)
14     dateedit.setDate(QDate.currentDate())
15     dateedit.setMinimumDate(QDate(1900, 1, 1))
16     dateedit.setMaximumDate(QDate(2100, 12, 31))
17     # dateedit.setDateRange(QDate(1900, 1, 1), QDate(2100, 12, 31))
18
19     vbox = QVBoxLayout()
20     vbox.addWidget(lbl)
21     vbox.addWidget(dateedit)
22     vbox.addStretch()
23
24     self.setLayout(vbox)
25
```

## Section 04 위젯

### □QDateEdit

- ex4\_16.py

```
26         self.setWindowTitle('QDateEdit')
27         self.setGeometry(300, 300, 300, 200)
28         self.show()
29
30     if __name__ == '__main__':
31         app = QApplication(sys.argv)
32         ex = MyApp()
33         app.exec()
```





## Section 04 위젯

### □QDateEdit

```
dateedit = QDateEdit(self)
dateedit.setDate(QDate.currentDate())
dateedit.setMinimumDate(QDate(1900, 1, 1))
dateedit.setMaximumDate(QDate(2100, 12, 31))
```

- QDateEdit 클래스를 이용해서 날짜 편집 위젯을 하나 생성한다.
- setDate 메서드에 QDate.currentDate()를 입력해서 프로그램이 실행될 때 현재 날짜로 표시되도록 한다.
- setMinimumDate와 setMaximumDate를 이용하면 사용자가 선택할 수 있는 날짜의 범위를 제한할 수 있다.
- 최소 날짜는 디폴트로 1752년 9월 14일로 설정되어 있고, 최대 날짜는 9999년 12월 31일로 설정되어 있다.
- 최소 날짜는 최소 100년 1월 1일 이상이어야 한다.

## Section 04 위젯

### □QDateEdit

```
# dateedit.setDateRange(QDate(1900, 1, 1), QDate(2100, 12, 31))
```

- **setDateRange** 메서드는 **setMinimumDate**와 **setMaximumDate**를 동시에 사용하는 것과 같다.

```
vbox = QVBoxLayout()  
vbox.addWidget(lbl)  
vbox.addWidget(dateedit)  
vbox.addStretch()
```

```
self.setLayout(vbox)
```

- 수직 박스 레이아웃을 이용해서 라벨과 날짜 편집 위젯을 수직으로 배치하고, 전체 위젯의 레이아웃으로 설정한다.

## Section 04 위젯

### □QTimeEdit

- QTimeEdit 위젯은 사용자에게 시간을 선택, 편집하도록 할 때 사용한다.
- 예제에서 QTimeEdit 객체를 하나 만들고, 현재 시간으로 표시되도록 설정해보자.
- ex\_4\_17.py

```
1      import sys
2      from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QTimeEdit, QVBoxLayout
3      from PyQt6.QtCore import Qtime
4
5      class MyApp(QWidget):
6          def __init__(self):
7              super().__init__()
8              self.initUI()
9
10         def initUI(self):
11             lbl = QLabel('QTimeEdit')
```

# Section 04 위젯

## □QTimeEdit

- ex\_4\_17.py

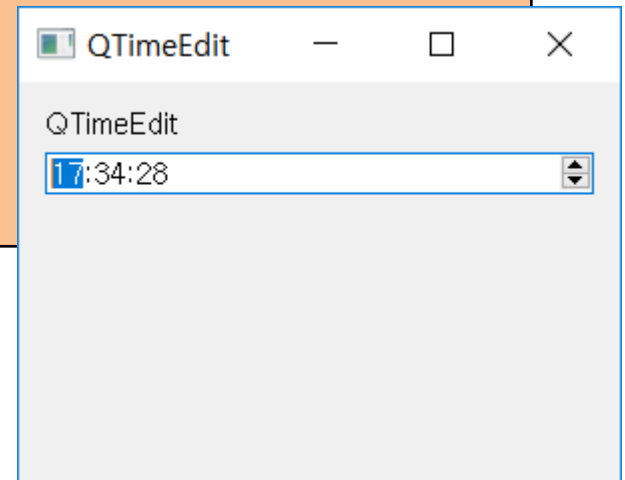
```
12
13     timeedit = QTimeEdit(self)
14     timeedit.setTime(QTime.currentTime())
15     timeedit.setTimeRange(QTime(3, 00, 00), QTime(23, 30, 00))
16     timeedit.setDisplayFormat('hh:mm:ss')
17
18     vbox = QVBoxLayout()
19     vbox.addWidget(lbl)
20     vbox.addWidget(timeedit)
21     vbox.addStretch()
22
23     self.setLayout(vbox)
24
25     self.setWindowTitle('QTimeEdit')
26     self.setGeometry(300, 300, 300, 200)
```

## Section 04 위젯

### □QTimeEdit

- ex\_4\_17.py

```
27         self.show()
28
29     if __name__ == '__main__':
30         app = QApplication(sys.argv)
31         ex = MyApp()
32         app.exec()
```



## Section 04 위젯

### □QTimeEdit

```
timeedit = QTimeEdit(self)
timeedit.setTime(QTime.currentTime())
timeedit.setTimeRange(QTime(3, 00, 00), QTime(23, 30, 00))
```

- QTimeEdit 클래스를 이용해서 시간 편집 위젯(timeedit)을 하나 생성한다.
- setTime 메서드에 QTime.currentTime()를 입력해서 프로그램이 실행될 때 현재 시간으로 표시되도록 한다.
- setTimeRange 이용하면 사용자가 선택할 수 있는 시간의 범위를 제한할 수 있다.
- 최소 시간은 디폴트로 00시 00분 00초 000밀리초로 설정되어 있고, 최대 시간은 23시 59분 59초 999밀리초로 설정되어 있다.

## Section 04 위젯

### □QTimeEdit

```
timeedit.setDisplayFormat('hh:mm:ss')
```

- **setDisplayFormat** 메서드를 이용해서 시간이 'hh:mm:ss'의 형식으로 표시되도록 설정했다.

```
vbox = QVBoxLayout()  
vbox.addWidget(lbl)  
vbox.addWidget(timeedit)  
vbox.addStretch()  
  
self.setLayout(vbox)
```

- 수직 박스 레이아웃을 이용해서 라벨과 시간 편집 위젯을 수직으로 배치하고, 전체 위젯의 레이아웃으로 설정한다.

## Section 04 위젯

### □ QDateTimeEdit

- QDateTimeEdit 위젯은 사용자에게 날짜와 시간을 선택, 편집하도록 할 때 사용한다.
- 예제에서 QDateTimeEdit 객체를 하나 만들고, 현재 날짜와 시간으로 표시되도록 설정해보자.
- ex\_4\_18.py

```
1 import sys
2 from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QDateTimeEdit, QVBoxLayout
3 from PyQt6.QtCore import QDateTime
4
5 class MyApp(QWidget):
6     def __init__(self):
7         super().__init__()
8         self.initUI()
9
10    def initUI(self):
```



# Section 04 위젯

## □ QDateTimeEdit

- ex\_4\_18.py

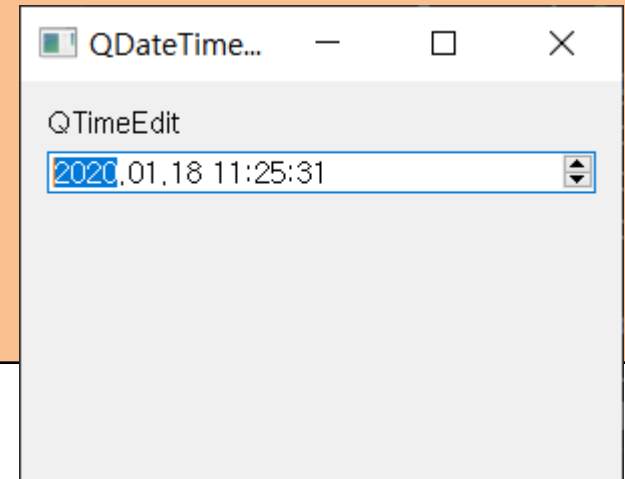
```
11     lbl = QLabel('QTimeEdit')
12
13     datetimeedit = QDateTimeEdit(self)
14     datetimeedit.setDateTime(QDateTime.currentDateTime())
15     datetimeedit.setDateTimeRange(QDateTime(1900, 1, 1, 00, 00, 00), QDateTime(2100, 1, 1,
16     00, 00, 00))
17
18     datetimeedit.setDisplayFormat('yyyy.MM.dd hh:mm:ss')
19
20     vbox = QVBoxLayout()
21     vbox.addWidget(lbl)
22     vbox.addWidget(datetimeedit)
23     vbox.addStretch()
24
25     self.setLayout(vbox)
```

# Section 04 위젯

## □ QDateTimeEdit

- ex\_4\_18.py

```
25     self.setWindowTitle('QDateTimeEdit')
26     self.setGeometry(300, 300, 300, 200)
27     self.show()
28
29 if __name__ == '__main__':
30     app = QApplication(sys.argv)
31     ex = MyApp()
32     app.exec()
```



## Section 04 위젯

---

### □ QDateTimeEdit

```
datetimeedit = QDateTimeEdit(self)
datetimeedit.setDateTime(QDateTime.currentDateTime())
datetimeedit.setDateTimeRange(QDateTime(1900, 1, 1, 00, 00, 00),
QDateTime(2100, 1, 1, 00, 00, 00))
```

- QDateTimeEdit 클래스를 이용해서 날짜, 시간 편집 위젯(datetimeedit)을 하나 생성한다.
- setDateTime 메서드에 QDateTime.currentDateTime()을 입력해서 프로그램이 실행될 때 현재 날짜와 시간으로 표시되도록 한다.
- setDateTimeRange를 이용하면 사용자가 선택할 수 있는 시간의 범위를 제한할 수 있다.

## Section 04 위젯

### □ QDateTimeEdit

```
datetimeedit.setDisplayFormat('yyyy.MM.dd hh:mm:ss')
```

- **setDisplayFormat** 메서드를 이용해서 시간이 'yyyy.MM.dd hh:mm:ss'의 형식으로 표시되도록 설정했다.

```
vbox = QVBoxLayout()  
vbox.addWidget(lbl)  
vbox.addWidget(datetimeedit)  
vbox.addStretch()  
  
self.setLayout(vbox)
```

- 수직 박스 레이아웃을 이용해서 라벨과 날짜, 시간 편집 위젯을 수직으로 배치하고, 전체 위젯의 레이아웃으로 설정한다.

## Section 04 위젯

---

### □QTextBrowser

- QTextBrowser 클래스는 하이퍼텍스트 네비게이션을 포함하는 리치 텍스트 (서식있는 텍스트) 브라우저를 제공한다.
- 이 클래스는 읽기 전용이며, QTextEdit의 확장형으로서 하이퍼텍스트 문서의 링크들을 사용할 수 있다.
- 편집 가능한 리치 텍스트 편집기를 사용하기 위해서는 QTextEdit을 사용해야 한다.
- 또한 하이퍼텍스트 네비게이션이 없는 텍스트 브라우저를 사용하기 위해서는 QTextEdit을 setReadOnly()를 사용해서 편집이 불가능하도록 한다.
- 짧은 리치 텍스트를 표시하기 위해서는 QLabel을 사용할 수 있다.

# Section 04 위젯

## □QTextBrowser

- ex\_4\_19.py

```
1 import sys
2 from PyQt6.QtWidgets import (QApplication, QWidget, QLineEdit, QTextBrowser, QPushButton,
3                               , QVBoxLayout)
4 class MyApp(QWidget):
5     def __init__(self):
6         super().__init__()
7         self.initUI()
8     def initUI(self):
9         self.le = QLineEdit()
10        self.le.returnPressed.connect(self.append_text)
11
12        self.tb = QTextBrowser()
13        self.tb.setAcceptRichText(True)
14        self.tb.setOpenExternalLinks(True)
```

# Section 04 위젯

## □QTextBrowser

- ex\_4\_19.py

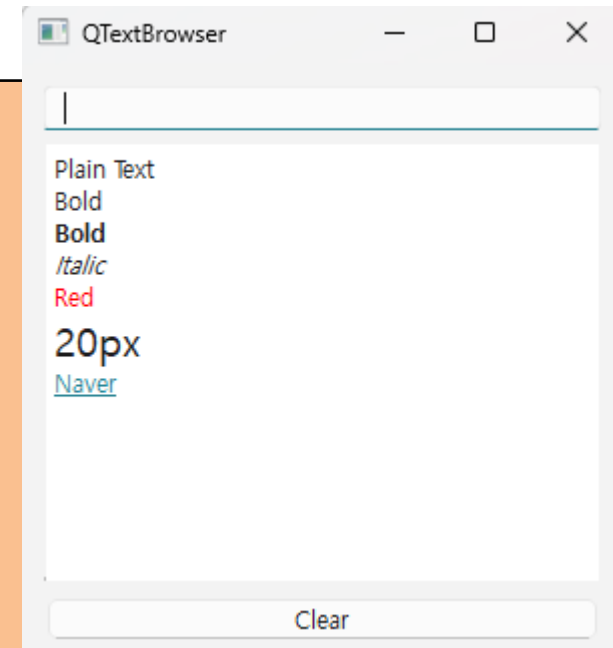
```
15
16     self.clear_btn = QPushButton('Clear')
17     self.clear_btn.pressed.connect(self.clear_text)
18
19     vbox = QVBoxLayout()
20     vbox.addWidget(self.le, 0)
21     vbox.addWidget(self.tb, 1)
22     vbox.addWidget(self.clear_btn, 2)
23
24     self.setLayout(vbox)
25
26     self.setWindowTitle('QTextBrowser')
27     self.setGeometry(300, 300, 300, 300)
28     self.show()
```

# Section 04 위젯

## □ QTextBrowser

- ex\_4\_19.py

```
29
30     def append_text(self):
31         text = self.le.text()
32         self.tb.append(text)
33         self.le.clear()
34
35     def clear_text(self):
36         self.tb.clear()
37
38 if __name__ == '__main__':
39     app = QApplication(sys.argv)
40     ex = MyApp()
41     app.exec()
```





## Section 04 위젯

### □QTextBrowser

```
self.le = QLineEdit()  
self.le.returnPressed.connect(self.append_text)
```

- 줄편집기 하나를 생성한다.
- Enter키를 누르면 append\_text 메서드가 호출된다.

```
self.tb = QTextBrowser()  
self.tb.setAcceptRichText(True)  
self.tb.setOpenExternalLinks(True)
```

- QTextBrowser() 클래스를 이용해서 텍스트 브라우저를 하나 생성한다.
- setAcceptRichText()를 True로 설정해주면, 서식 있는 텍스트 (Rich text)를 사용할 수 있다.
- 디폴트로 True이기 때문에 없어도 되는 부분이다.
- setOpenExternalLinks()를 True로 설정해주면, 외부 링크로의 연결이

## Section 04 위젯

### □QTextBrowser

```
self.clear_btn = QPushButton('Clear')  
self.clear_btn.pressed.connect(self.clear_text)
```

- **clear\_btn을 클릭하면, clear\_text 메서드가 호출된다.**

```
def append_text(self):  
  
    text = self.le.text()  
    self.tb.append(text)  
    self.le.clear()
```

- **append\_text 메서드는 줄편집기에 작성된 텍스트 (self.le.text())를 텍스트 브라우저 (self.tb)에 append 해주는 기능을 한다.**
- **텍스트가 텍스트 브라우저에 추가되면, clear 메서드를 이용해서 줄편집기에 있던 텍스트는 없애준다.**

## Section 04 위젯

### □QTextBrowser

```
def clear_text(self):  
  
    self.tb.clear()
```

- **clear\_text** 메서드가 호출되면, **clear** 메서드를 이용해서 텍스트 브라우저 [self.tb]에 있던 텍스트를 없애준다.

```
Plain Text  
<b>Bold</b>  
<i>Italic</i>  
<p style="color: red">Red</p>  
<p style="font-size: 20px">20px</p>  
<a href="https://www.naver.com">Naver</a>
```

- **줄편집기**에 위의 텍스트를 순서대로 입력하고 엔터키를 이용해서 텍스트 브라우저에 추가한다.

## Section 04 위젯

### □QTextEdit

- QTextEdit 클래스는 플레인 텍스트 (plain text)와 리치 텍스트 (rich text)를 모두 편집하고 표시할 수 있는 편집기를 제공한다.
- 두 개의 라벨과 하나의 텍스트 편집기를 이용해서 단어수를 표시하는 간단한 프로그램을 만들어 보자.
- ex\_4\_20.py

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QTextEdit, QVBoxLayout
3
4  class MyApp(QWidget):
5      def __init__(self):
6          super().__init__()
7          self.initUI()
8
9      def initUI(self):
10         self.lbl1 = QLabel('Enter your sentence:')
```

# Section 04 위젯

## □QTextEdit

- ex\_4\_21.py

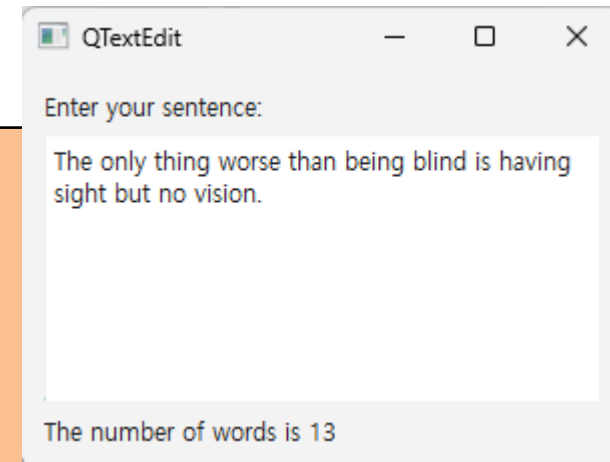
```
11         self.te = QTextEdit()
12
13         self.te.setAcceptRichText(False)
14
15         self.lbl2 = QLabel('The number of words is 0')
16
17         self.te.textChanged.connect(self.text_changed)
18
19         vbox = QVBoxLayout()
20         vbox.addWidget(self.lbl1)
21         vbox.addWidget(self.te)
22         vbox.addWidget(self.lbl2)
23         vbox.addStretch()
24
25         self.setLayout(vbox)
```

# Section 04 위젯

## □ QTextEdit

- ex\_4\_21.py

```
26         self.setGeometry(300, 300, 300, 200)
27         self.show()
28
29         def text_changed(self):
30             text = self.te.toPlainText()
31             self.lbl2.setText('The number of words is ' + str(len(text.split())))
32
33     if __name__ == '__main__':
34         app = QApplication(sys.argv)
35         ex = MyApp()
36         app.exec()
```



## Section 04 위젯

---

### □QTextEdit

```
self.lbl1 = QLabel('Enter your sentence:')  
self.te = QTextEdit()  
self.te.setAcceptRichText(False)  
self.lbl2 = QLabel('The number of words is 0')
```

- QTextEdit() 클래스를 이용해서 텍스트 편집기 하나를 생성한다.
- setAcceptRichText를 False로 하면, 모두 플레인 텍스트로 인식한다.
- 아래의 라벨은 단어수를 표시한다.

```
self.te.textChanged.connect(self.text_changed)
```

- 텍스트 편집기의 텍스트가 수정될 때마다 text\_changed 메서드가 호출된다.

## Section 04 위젯

---

### □QTextEdit

```
vbox = QVBoxLayout()  
vbox.addWidget(self.lbl1)  
vbox.addWidget(self.te)  
vbox.addWidget(self.lbl2)  
vbox.addStretch()  
  
self.setLayout(vbox)
```

- 수직 박스 레이아웃을 이용해서, 두 개의 라벨과 하나의 텍스트 편집기를 수직 방향으로 배치한다.



## Section 04 위젯

---

### □QTextEdit

```
def text_changed(self):  
  
    text = self.te.toPlainText()  
    self.lbl2.setText('The number of words is ' + str(len(text.split())))
```

- **text\_changed** 메서드가 호출되면, **toPlainText()** 메서드를 이용해서 텍스트 편집기 (**self.te**)에 있던 텍스트를 **text** 변수에 저장한다.
- **split()**은 문자열의 단어들을 리스트 형태로 바꿔준다.
- **len(text.split())**은 **text**의 단어수이다.
- **setText()**를 이용해서 두 번째 라벨에 단어수를 표시한다.

## Section 04 위젯

### □QTableWidget

- QTableWidget 클래스는 테이블 형태로 항목을 배치하고 다루도록 한다.
- 이 절에서는 PyQt6 위젯에서 테이블을 만들어 항목을 배치하고 수정하는 방법을 소개한다.
- ex\_4\_21.py

```
1      import sys
2
3      from PyQt6.QtWidgets import *
4
5      class MyApp(QWidget):
6          def __init__(self):
7              super().__init__()
8              self.initUI()
9
10         def initUI(self):
11             self.tableWidget = QTableWidget()
12             self.tableWidget.setRowCount(20)
```

# Section 04 위젯

## □QTableWidget

- ex\_4\_21.py

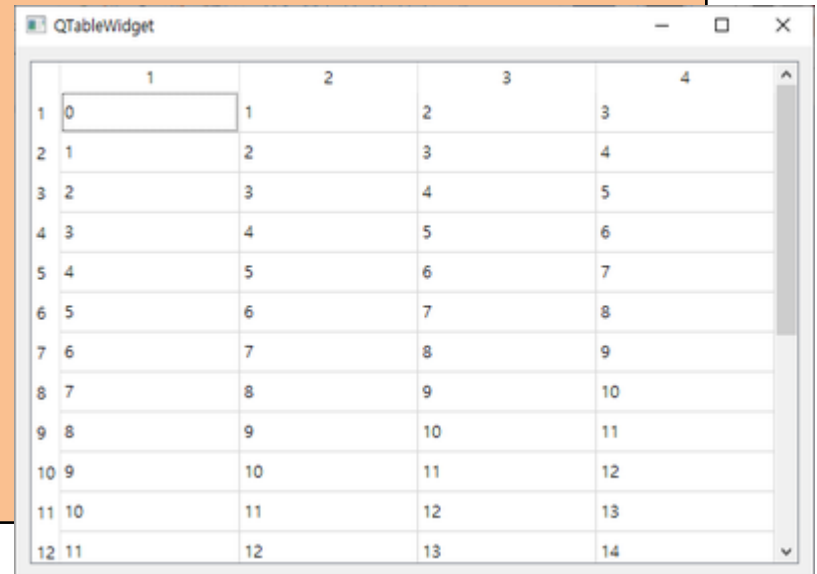
```
12         self.tableWidget.setColumnCount(4)
13
14         self.tableWidget.setEditTriggers(QAbstractItemView.EditTrigger.NoEditTriggers)
15         # self.tableWidget.setEditTriggers(QAbstractItemView.DoubleClicked)
16         # self.tableWidget.setEditTriggers(QAbstractItemView.AllEditTriggers)
17
18         self.tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode
19         .Stretch)
20         # self.tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeTo
21         Contents)
22
23         for i in range(20):
24             for j in range(4):
25                 self.tableWidget.setItem(i, j, QTableWidgetItem(str(i+j)))
```

## Section 04 위젯

### □QTableWidget

- ex\_4\_21.py

```
25         layout = QVBoxLayout()
26         layout.addWidget(self.tableWidget)
27         self.setLayout(layout)
28
29         self.setWindowTitle('QTableWidget')
30         self.setGeometry(300, 100, 600, 400)
31         self.show()
32
33     if __name__ == '__main__':
34         app = QApplication(sys.argv)
35         ex = MyApp()
36         app.exec()
```



The screenshot shows a Qt application window titled "QTableWidget". Inside the window is a table with 12 rows and 4 columns. The columns are indexed 1 to 4, and the rows are indexed 1 to 12. The table contains a sequence of numbers from 0 to 14, starting with 0 in the first cell (row 1, column 1) and increasing by 1 for each subsequent cell in row-major order. A vertical scrollbar is visible on the right side of the table, indicating that the content can be scrolled.

	1	2	3	4
1	0	1	2	3
2	1	2	3	4
3	2	3	4	5
4	3	4	5	6
5	4	5	6	7
6	5	6	7	8
7	6	7	8	9
8	7	8	9	10
9	8	9	10	11
10	9	10	11	12
11	10	11	12	13
12	11	12	13	14

## Section 04 위젯

---

### □QTableWidget

```
self.tableWidget = QTableWidget()  
self.tableWidget.setRowCount(20)  
self.tableWidget.setColumnCount(4)
```

- QTableWidget 클래스를 사용해서 테이블 위젯을 하나 생성한다.
- setRowCount() 메서드는 테이블의 행 (Row)의 개수를 지정한다.
- setColumnCount() 메서드는 테이블의 열 (Column)의 개수를 지정한다.

## Section 04 위젯

---

### □QTableWidget

```
self.tableWidget.setEditTriggers(QAbstractItemView.NoEditTriggers)
# self.tableWidget.setEditTriggers(QAbstractItemView.DoubleClicked)
# self.tableWidget.setEditTriggers(QAbstractItemView.AllEditTriggers)
```

- **setEditTriggers()** 메서드는 테이블의 항목을 편집 가능하도록 하는 액션을 지정한다.
- **QAbstractItemView.NoEditTriggers**로 지정하면 편집을 할 수 없다.
- **QAbstractItemView.DoubleClicked**로 지정하면 칸을 더블클릭했을 때 편집이 가능하다.
- **QAbstractItemView.AllEditTriggers**로 지정하면 클릭, 더블클릭 등 모든 액션에 대해 편집이 가능하도록 한다.

## Section 04 위젯

---

### □QTableWidget

```
self.tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)  
# self.tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeToContents)
```

- **horizontalHeader()**는 수평 헤더를 반환한다.
- **setSectionResizeMode()** 메서드는 헤더의 크기를 조절하는 방식을 지정한다.
- **QHeaderView.Stretch**는 헤더의 폭이 위젯의 폭에 맞춰지도록 한다.
- **QHeaderView.ResizeToContents**는 헤더의 폭이 항목 값의 폭에 맞춰지도록 한다.

## Section 04 위젯

---

### □QTableWidget

```
for i in range(20):  
    for j in range(4):  
        self.tableWidget.setItem(i, j, QTableWidgetItem(str(i+j)))
```

- `setItem(row, column, value)` 메서드는 테이블 항목의 값을 지정한다.
- 순서대로 행과 열의 번호, 그리고 값을 입력한다.



## Section 05 다이얼로그(Dialog)

---

### □개요

- 다이얼로그는 대화창이라고도 부르며 GUI 프로그래밍에서 없어서는 안될 요소이다.
- 이름에서 알 수 있듯이 사용자가 어플리케이션 안에서 어플리케이션과 '대화' 하는데 사용된다.
- 다시 말해서 다이얼로그는 사용자가 데이터를 입력, 수정하거나, 어플리케이션의 설정을 변경하는 등의 작업을 하는데 사용된다.
- 예제를 통해 다양한 다이얼로그를 사용하는 방법에 대해 알아보자.

## Section 05 다이얼로그(Dialog)

---

### □QInputDialog

- 입력 다이얼로그 (QInputDialog)는 사용자가 간단한 값을 입력할 때 사용하는 다이얼로그이다.
- 입력값은 숫자, 문자열, 리스트에서 선택한 항목 등이 될 수 있다.  
[QInputDialog 공식 문서 참고]
- 입력값의 형태에 따라 아래와 같이 다섯 개의 유용한 함수가 제공된다.
  - getText()
  - getMultiLineText()
  - getInt()
  - getDouble()
  - getItem()
- 예제에서는 getText() 메서드를 사용한다.

# Section 05 다이얼로그(Dialog)

## □ QDialog

- ex5\_1.py

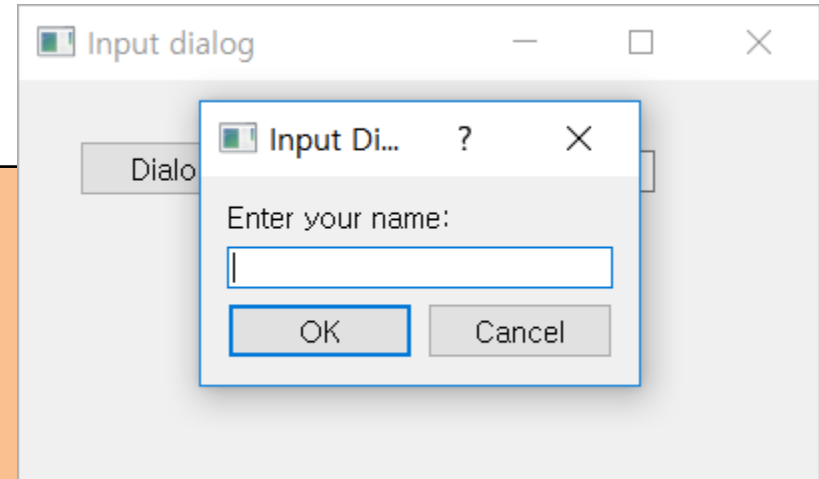
```
1  import sys
2  from PyQt6.QtWidgets import (QApplication, QWidget, QPushButton, QLineEdit, QDialog)
3
4  class MyApp(QWidget):
5      def __init__(self):
6          super().__init__()
7          self.initUI()
8
9      def initUI(self):
10         self.btn = QPushButton('Dialog', self)
11         self.btn.move(30, 30)
12         self.btn.clicked.connect(self.showDialog)
13
14         self.le = QLineEdit(self)
15         self.le.move(120, 35)
```

# Section 05 다이얼로그(Dialog)

## ❑ QDialog

### • ex5\_1.py

```
16
17     self.setWindowTitle('Input dialog')
18     self.setGeometry(300, 300, 300, 200)
19     self.show()
20
21     def showDialog(self):
22         text, ok = QDialog.getText(self, 'Input Dialog', 'Enter your name:')
23
24         if ok:
25             self.le.setText(str(text))
26
27 if __name__ == '__main__':
28     app = QApplication(sys.argv)
29     ex = MyApp()
30     app.exec()
```



## Section 05 다이얼로그(Dialog)

### □QInputDialog

```
text, ok = QInputDialog.getText(self, 'Input Dialog', 'Enter your name:')
```

- 이 코드를 통해 입력 대화창이 나타난다.
- 두 번째 매개변수는 대화창의 타이틀, 세 번째 매개변수는 대화창 안에 보여질 메시지를 의미한다.
- 입력 대화창은 입력한 텍스트와 불 (Boolean) 값을 반환한다.
- 텍스트를 입력한 후 'OK' 버튼을 누르면 불 값은 True, 'Cancel' 버튼을 누르면 불 값은 False가 된다.

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

- 입력한 값을 setText() 메서드를 통해 줄 편집 위젯에 표시되도록 한다.

## Section 05 다이얼로그(Dialog)

### □ QColorDialog

- 컬러 다이얼로그 (QColorDialog)는 색상을 선택할 수 있는 다이얼로그이다.
- ex\_5\_2.py

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget, QPushButton, QFrame, QColorDialog
3  from PyQt6.QtGui import QColor
4
5  class MyApp(QWidget):
6      def __init__(self):
7          super().__init__()
8          self.initUI()
9
10     def initUI(self):
11         col = QColor(0, 0, 0)
12
13         self.btn = QPushButton('Dialog', self)
```

## Section 05 다이얼로그(Dialog)

### □ QColorDialog

- 컬러 다이얼로그 (QColorDialog)는 색상을 선택할 수 있는 다이얼로그이다.
- ex\_5\_2.py

```
14         self.btn.move(30, 30)
15
16         self.btn.clicked.connect(self.showDialog)
17
18         self.frm = QFrame(self)
19         self.frm.setStyleSheet('QWidget { background-color: %s }' % col.name())
20         self.frm.setGeometry(130, 35, 100, 100)
21
22         self.setWindowTitle('Color Dialog')
23         self.setGeometry(300, 300, 250, 180)
24         self.show()
```

## Section 05 다이얼로그(Dialog)

### □ QColorDialog

- 컬러 다이얼로그 (QColorDialog)는 색상을 선택할 수 있는 다이얼로그이다.
- ex\_5\_2.py

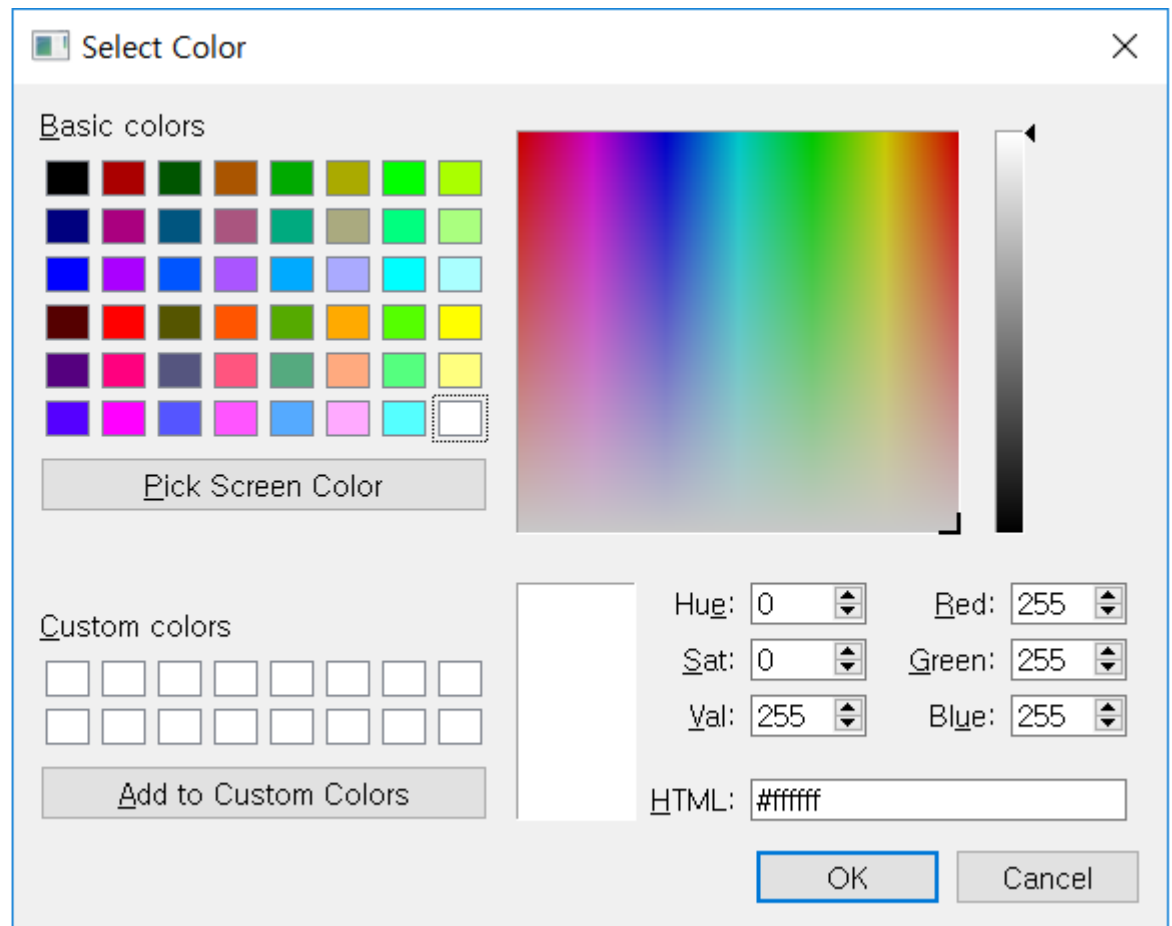
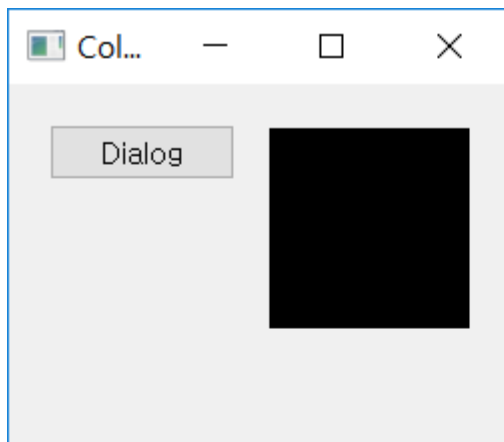
```
25     def showDialog(self):
26         col = QColorDialog.getColor()
27
28         if col.isValid():
29             self.frm.setStyleSheet('QWidget { background-color: %s }' % col.name())
30
31 if __name__ == '__main__':
32     app = QApplication(sys.argv)
33     ex = MyApp()
34     app.exec()
```



## Section 05 다이얼로그(Dialog)

### □ QColorDialog

- ex\_5\_2.py



## Section 05 다이얼로그(Dialog)

### □ QColorDialog

```
col = QColor(0, 0, 0)
```

- QColor를 사용해서 배경색인 검정색을 생성한다.

```
col = QColorDialog.getColor()
```

- QColorDialog 클래스의 getColor() 메서드는 컬러 다이얼로그 창을 띄우고 사용자가 색상을 선택하도록 한다.
- 그리고 선택한 색상을 QColor 클래스의 형태로 반환한다.

```
if col.isValid():  
    self.frm.setStyleSheet('QWidget { background-color: %s }' % col.name())
```

- 색상을 선택하고 'OK' 버튼을 눌렀다면, col.isValid()의 불 값이 True이고, 'Cancel' 버튼을 눌렀다면, 불 값이 False가 된다.

## Section 05 다이얼로그(Dialog)

### □ QFontDialog

- 폰트 다이얼로그 (QFontDialog)는 폰트를 선택할 수 있게 해주는 다이얼로그이다.
- ex\_5\_3.py

```
1  import sys
2
3  from PyQt6.QtWidgets import (QApplication, QWidget, QVBoxLayout
4  , QPushButton, QSizePolicy, QLabel, QFontDialog)
5
6  class MyApp(QWidget):
7
8      def __init__(self):
9          super().__init__()
10         self.initUI()
11
12     def initUI(self):
13         btn = QPushButton('Dialog', self)
14         btn.setSizePolicy(QSizePolicy(QSizePolicy.Policy.Fixed, QSizePolicy.Policy.Fixed))
15         btn.move(20, 20)
```

# Section 05 다이얼로그(Dialog)

## □ QFontDialog

- ex\_5\_4.py

```
13         btn.clicked.connect(self.showDialog)
14
15         vbox = QVBoxLayout()
16         vbox.addWidget(btn)
17
18         self.lbl = QLabel('The quick brown fox jumps over the lazy dog', self)
19         self.lbl.move(130, 20)
20
21         vbox.addWidget(self.lbl)
22         self.setLayout(vbox)
23
24         self.setWindowTitle('Font Dialog')
25         self.setGeometry(300, 300, 250, 180)
26         self.show()
```

## Section 05 다이얼로그(Dialog)

### □ QFontDialog

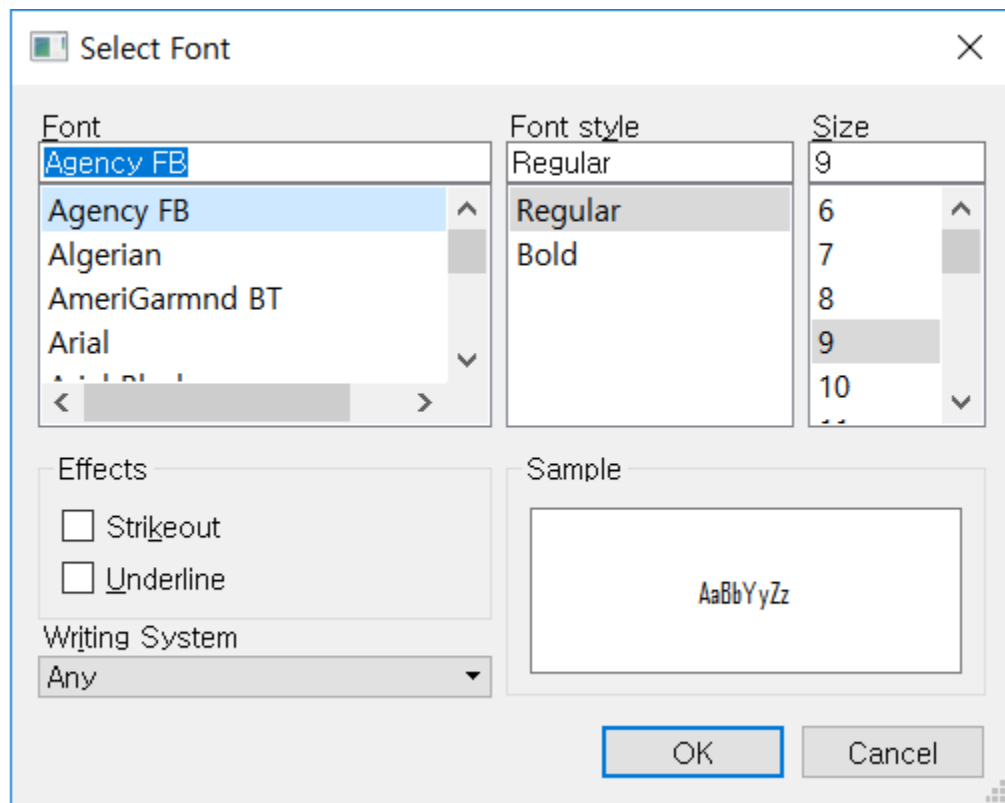
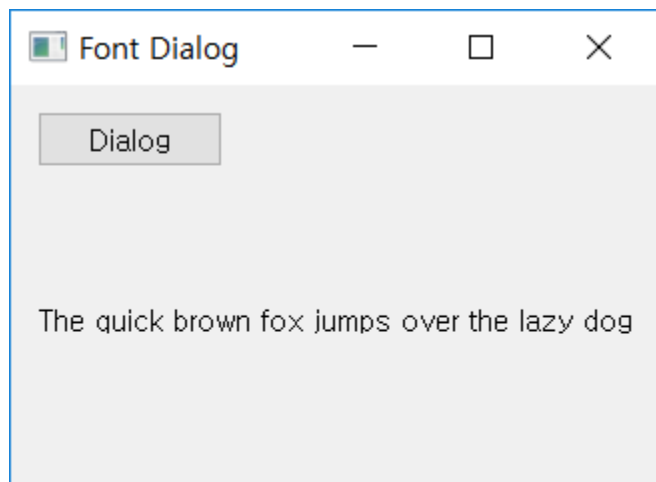
- ex\_5\_4.py

```
27
28     def showDialog(self):
29         font, ok = QFontDialog.getFont()
30
31         if ok:
32             self.lbl.setFont(font)
33
34 if __name__ == '__main__':
35     app = QApplication(sys.argv)
36     ex = MyApp()
37     app.exec()
```

## Section 05 다이얼로그(Dialog)

### □ QFontDialog

- ex\_5\_4.py



## Section 05 다이얼로그(Dialog)

### □ QFontDialog

```
font, ok = QFontDialog.getFont()
```

- 폰트 다이얼로그를 띄우고, `getFont()` 메서드를 사용해서 선택한 폰트와 불값을 반환받는다.
- 앞의 예제와 마찬가지로 'OK' 버튼을 클릭하면 `True`를, 'Cancel' 버튼을 클릭하면 `False`를 반환한다.

```
if ok:  
    self.lbl.setFont(font)
```

- `setFont()` 메서드를 사용해서 선택한 폰트를 라벨의 폰트로 설정해준다.

## Section 05 다이얼로그(Dialog)

### □QFileDialog

- QFileDialog는 사용자가 파일 또는 경로를 선택할 수 있도록 하는 다이얼로그이다.
- 사용자는 선택한 파일을 열어서 수정하거나 저장할 수 있다.
- ex\_5\_4.py

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QMainWindow, QTextEdit, QFileDialog
3  from PyQt6.QtGui import QIcon, QAction
4
5  class MyApp(QWidget):
6      def __init__(self):
7          super().__init__()
8          self.initUI()
9
10     def initUI(self):
12         self.textEdit = QTextEdit()
```



# Section 05 다이얼로그(Dialog)

## □QFileDialog

- ex\_5\_4.py

```
13         self.setCentralWidget(self.textEdit)
14
15         self.statusBar()
16
17         openFile = QAction(QIcon('open.png'), 'Open', self)
18         openFile.setShortcut('Ctrl+O')
19         openFile.setStatusTip('Open New File')
20         openFile.triggered.connect(self.showDialog)
21
22         menubar = self.menuBar()
23         menubar.setNativeMenuBar(False)
24         fileMenu = menubar.addMenu('&File')
25         fileMenu.addAction(openFile)
26
27         self.setWindowTitle('File Dialog')
28         self.setGeometry(300, 300, 300, 200)
```

## Section 05 다이얼로그(Dialog)

### □QFileDialog

- ex\_5\_4.py

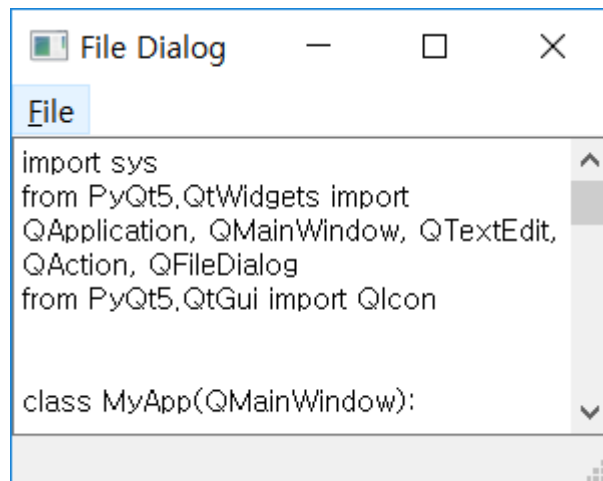
```
28         self.show()
29
30     def showDialog(self):
31         fname = QFileDialog.getOpenFileName(self, 'Open file', './')
32
33         if fname[0]:
34             f = open(fname[0], 'r')
35
36             with f:
37                 data = f.read()
38                 self.textEdit.setText(data)
39
```

## Section 05 다이얼로그(Dialog)

### □QFileDialog

- ex\_5\_4.py

```
40  if __name__ == '__main__':  
41      app = QApplication(sys.argv)  
42      ex = MyApp()  
43      app.exec()
```



## Section 05 다이얼로그(Dialog)

### □QFileDialog

```
fname = QFileDialog.getOpenFileName(self, 'Open file', './')
```

- QFileDialog를 띄우고, getOpenFileName() 메서드를 사용해서 파일을 선택한다.
- 세 번째 매개변수를 통해 기본 경로를 설정할 수 있습니다. 또한 기본적으로 모든 파일( \*)을 열도록 되어있다.

```
if fname[0]:  
    f = open(fname[0], 'r')  
  
    with f:  
        data = f.read()  
        self.textEdit.setText(data)
```

- 선택한 파일을 읽어서, setText() 메서드를 통해 텍스트 편집 위젯에 불러온다.

## Section 05 다이얼로그(Dialog)

### □ QMessageBox

- QMessageBox 클래스는 사용자에게 정보를 제공하거나 질문과 대답을 할 수 있는 대화창을 제공한다.
- 흔히 어떤 동작에 대해 확인이 필요한 경우에 메시지 박스를 사용한다.
- 메시지 박스에서는 사용자에게 상황을 설명하는 기본 텍스트를 표시한다.
- 그 다음 정보를 전달하거나 사용자의 의사를 묻는 텍스트를 표시할 수 있다.
- 마지막으로 더욱 자세히 상황을 설명하기 위한 세부적인 텍스트를 표시할 수 있다.
- 이러한 각각의 텍스트를 표시하기 위해 `setText()/setInformativeText()/setDetailedText()` 메서드를 사용할 수 있다.



## Section 05 다이얼로그(Dialog)

### □ QMessageBox

- ex\_5\_5.py

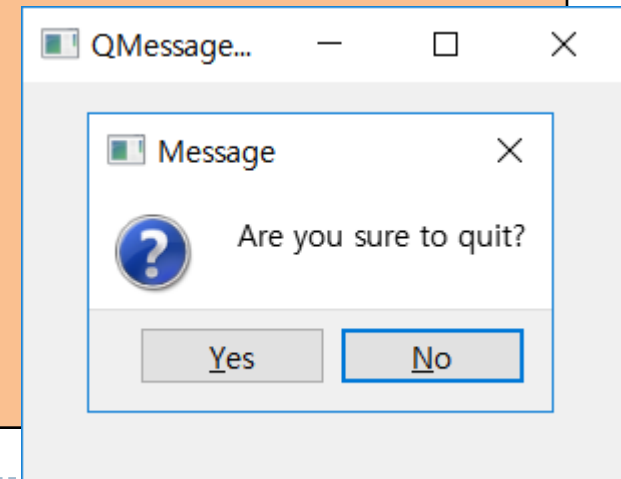
```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget, QMessageBox
3
4  class MyApp(QWidget):
5      def __init__(self):
6          super().__init__()
7          self.initUI()
8
9      def initUI(self):
10         self.setWindowTitle('QMessageBox')
11
12         self.setGeometry(300, 300, 300, 200)
13         self.show()
14
```

## Section 05 다이얼로그(Dialog)

### □ QMessageBox

- ex\_5\_5.py

```
15     def closeEvent(self, event):
16         reply = QMessageBox.question(self, 'Message', 'Are you sure to quit?',
                                       QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No,
                                       QMessageBox.StandardButton.No)
17         if reply == QMessageBox.StandardButton.Yes:
18             event.accept()
19         else:
20             event.ignore()
21
22     if __name__ == '__main__':
23         app = QApplication(sys.argv)
24         ex = MyApp()
25         app.exec()
```



## Section 05 다이얼로그(Dialog)

### □ QMessageBox

- QWidget을 종료할 때, QCloseEvent가 생성되어 위젯에 전달된다.
- 위젯의 동작을 수정하기 위해 closeEvent() 이벤트 핸들러를 수정해야 한다.

```
reply = QMessageBox.question(self, 'Message', 'Are you sure to quit?',  
                             QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
```

- 두번째 매개변수는 타이틀바에 나타날 문자열 ('Message'), 세번째 매개변수는 대화창에 나타날 문자열 ('Are you sure to quit?')을 입력한다.
- 네 번째에는 대화창에 보여질 버튼의 종류를 입력하고, 마지막으로 디폴트로 선택될 버튼을 설정해준다.
- QMessageBox.No로 설정할 경우, 메시지 박스가 나타났을 때 'No' 버튼이 선택되어 있다.
- 반환값은 reply 변수에 저장된다.



## Section 05 다이얼로그(Dialog)

---

### □ QMessageBox

```
if reply == QMessageBox.Yes:  
    event.accept()  
else:  
    event.ignore()
```

- 'Yes' 버튼을 클릭했을 경우, 이벤트를 받아들이고 위젯을 종료한다.
- 'No' 버튼을 클릭하면, 이벤트를 무시하고 위젯을 종료하지 않는다.

# Section 06 시그널과 슬롯(Signal & Slot)

---

## □개요

- PyQt에서는 이벤트 처리에 있어서 시그널과 슬롯이라는 독특한 메커니즘을 사용한다.
- 간단한 예제들을 통해 시그널과 슬롯의 연결에 대해 알아보자.
- 순서는 아래와 같다.
  - 연결하기
  - 이벤트 핸들러 만들기
  - 이벤트 핸들러 재구성하기
  - 이벤트 핸들러 재구성하기2
  - 사용자 정의 시그널

# Section 06 시그널과 슬롯(Signal & Slot)

## □연결하기

- 다이얼 위젯으로 조절한 값을 화면에 출력하는 프로그램을 만들어 보자.
- 다이얼의 값이 변할 때 발생하는 시그널이 LCD 화면에 숫자를 표시하는 슬롯과 연결된다.
- ex\_6\_1.py

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget, QLCDNumber, QDial, QVBoxLayout
3
4  class MyApp(QWidget):
5      def __init__(self):
6          super().__init__()
7          self.initUI()
8
9      def initUI(self):
10         lcd = QLCDNumber(self)
11
12         dial = QDial(self)
```

## Section 06 시그널과 슬롯(Signal & Slot)

### □연결하기

- ex\_6\_1.py

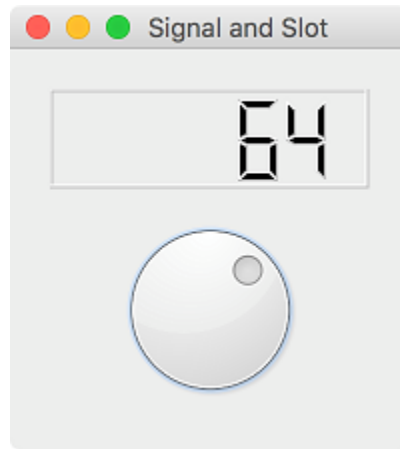
```
13
14     vbox = QVBoxLayout()
15     vbox.addWidget(lcd)
16     vbox.addWidget(dial)
17     self.setLayout(vbox)
18
19     dial.valueChanged.connect(lcd.display)
20
21     self.setWindowTitle('Signal and Slot')
22     self.setGeometry(300, 300, 200, 200)
23     self.show()
24
```

## Section 06 시그널과 슬롯(Signal & Slot)

### □연결하기

- ex\_6\_1.py

```
25 if __name__ == '__main__':  
26     app = QApplication(sys.argv)  
27     ex = MyApp()  
28     app.exec()
```



## Section 06 시그널과 슬롯(Signal & Slot)

### □연결하기

```
lcd = QLCDNumber(self)
dial = QDial(self)
```

- **QLCDNumber** 위젯은 LCD 화면과 같이 숫자를 표시한다.
- **QDial**은 회전해서 값을 조절하는 위젯이다.

```
vbox = QVBoxLayout()
vbox.addWidget(lcd)
vbox.addWidget(dial)
self.setLayout(vbox)
```

- 수직 박스 레이아웃(박스 레이아웃 참고)을 하나 만들어서 **QLCDNumber**와 **QDial** 위젯을 넣어준다.
- 그리고 **MyApp** 위젯의 레이아웃으로 설정한다.

## Section 06 시그널과 슬롯(Signal & Slot)

### □연결하기

```
dial.valueChanged.connect(lcd.display)
```

- QDial 위젯은 몇 가지 시그널을 갖고 있다(QSlider, QDial 참고).
- 여기서는 valueChanged 시그널을 lcd의 display 슬롯에 연결한다.
- Display 슬롯은 숫자를 받아서 QLCDNumber 위젯에 표시하는 역할을 한다.
- 여기서 시그널을 보내는 객체인 송신자 (sender)는 dial, 시그널을 받는 객체인 수신자 (receiver)는 lcd이다.
- 슬롯 (slot)은 시그널에 어떻게 반응할지를 구현한 메서드이다.

# Section 06 시그널과 슬롯(Signal & Slot)

## □이벤트 핸들러 만들기

- PyQt에서 이벤트(시그널) 처리를 할 때 사용되는 함수를 이벤트 핸들러 [슬롯]라고 한다.
- 'Big', 'Small' 버튼을 클릭(시그널이 발생)했을 때, 창의 크기가 바뀌도록 하는 함수(슬롯)를 정의해보자.
- ex\_6\_2.py

```
1  import sys
2
3  from PyQt6.QtWidgets import (QApplication, QWidget, QLCDNumber, QDial, QPushButton,
4                                QVBoxLayout, QHBoxLayout)
5
6  class MyApp(QWidget):
7      def __init__(self):
8          super().__init__()
9          self.initUI()
10
11      def initUI(self):
12          lcd = QLCDNumber(self)
```



# Section 06 시그널과 슬롯(Signal & Slot)

## □이벤트 핸들러 만들기

- ex\_6\_3.py

```
11     dial = QDial(self)
12     btn1 = QPushButton('Big', self)
13     btn2 = QPushButton('Small', self)
14
15     hbox = QHBoxLayout()
16     hbox.addWidget(btn1)
17     hbox.addWidget(btn2)
18
19     vbox = QVBoxLayout()
20     vbox.addWidget(lcd)
21     vbox.addWidget(dial)
22     vbox.addLayout(hbox)
23     self.setLayout(vbox)
24
25     dial.valueChanged.connect(lcd.display)
```

# Section 06 시그널과 슬롯(Signal & Slot)

## □ 이벤트 핸들러 만들기

- ex\_6\_3.py

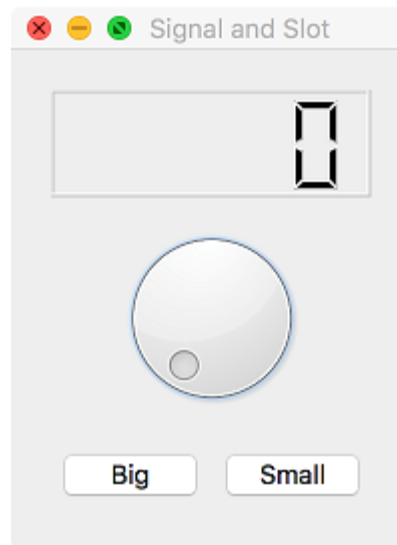
```
26         btn1.clicked.connect(self.resizeBig)
27         btn2.clicked.connect(self.resizeSmall)
28
29         self.setWindowTitle('Signal and Slot')
30         self.setGeometry(200, 200, 200, 250)
31         self.show()
32
33         def resizeBig(self):
34             self.resize(400, 500)
35
36         def resizeSmall(self):
37             self.resize(200, 250)
38
```

# Section 06 시그널과 슬롯(Signal & Slot)

## □이벤트 핸들러 만들기

- ex\_6\_3.py

```
39  if __name__ == '__main__':  
40      app = QApplication(sys.argv)  
41      ex = MyApp()  
42      app.exec()
```



## Section 06 시그널과 슬롯(Signal & Slot)

### □이벤트 핸들러 만들기

```
btn1.clicked.connect(self.resizeBig)
btn2.clicked.connect(self.resizeSmall)
```

- btn1과 btn2는 각각 resizeBig, resizeSmall 슬롯에 연결되어 있다.

```
def resizeBig(self):
    self.resize(400, 500)

def resizeSmall(self):
    self.resize(200, 250)
```

- resizeBig() 메서드는 화면 크기를 가로 400px, 세로 500px로 확대, resizeSmall() 메서드는 화면 크기를 가로 200px, 세로 250px로 축소하는 기능을 가지게 된다.

## Section 06 시그널과 슬롯(Signal & Slot)

### □이벤트 핸들러 재구성하기

- 아래와 같이 자주 쓰이는 이벤트 핸들러는 이미 만들어져 있는 경우가 많다.

이벤트 핸들러	설명
keyPressEvent	키보드를 눌렀을 때 동작한다.
keyReleaseEvent	키보드를 눌렀다가 떼 때 동작한다.
mouseDoubleClickEvent	마우스를 더블클릭할 때 동작한다.
mouseMoveEvent	마우스를 움직일 때 동작한다.
mousePressEvent	마우스를 누를 때 동작한다.
mouseReleaseEvent	마우스를 눌렀다가 떼 때 동작한다.
moveEvent	위젯이 이동할 때 동작한다.
resizeEvent	위젯의 크기를 변경할 때 동작한다.

# Section 06 시그널과 슬롯(Signal & Slot)

## □이벤트 핸들러 재구성하기

- **keyPressEvent** 이벤트 핸들러를 수정해서, 특정 키를 눌렀을 때 위젯을 종료하거나 최대화, 보통 크기로 조절하는 기능을 구현해보자.
- **ex\_6\_3.py**

```
1  import sys
2  from PyQt6.QtCore import Qt
3  from PyQt6.QtWidgets import QApplication, QWidget
4
5  class MyApp(QWidget):
6      def __init__(self):
7          super().__init__()
8          self.initUI()
9
10     def initUI(self):
11         self.setWindowTitle('Reimplementing event handler')
12         self.setGeometry(300, 300, 300, 200)
```

# Section 06 시그널과 슬롯(Signal & Slot)

## □이벤트 핸들러 재구성하기

- ex\_6\_3.py

```
13         self.show()
14
15     def keyPressEvent(self, e):
16         if e.key() == Qt.Key.Key_Escape:
17             self.close()
18         elif e.key() == Qt.Key.Key_F:
19             self.showFullScreen()
20         elif e.key() == Qt.Key.Key_N:
21             self.showNormal()
22
23 if __name__ == '__main__':
24     app = QApplication(sys.argv)
25     ex = MyApp()
26     app.exec()
```

## Section 06 시그널과 슬롯(Signal & Slot)

### □이벤트 핸들러 재구성하기

- 위 예제는 'esc', 'F', 'N' 키를 클릭하면 창이 종료되거나 최대화, 보통 크기가 되도록 이벤트 핸들러를 재구성했다.

```
def keyPressEvent(self, e):  
  
    if e.key() == Qt.Key_Escape:  
        self.close()  
    elif e.key() == Qt.Key_F:  
        self.showFullScreen()  
    elif e.key() == Qt.Key_N:  
        self.showNormal()
```

- keyPressEvent 이벤트 핸들러는 키보드의 이벤트를 입력으로 받는다.
- e.key()는 어떤 키를 누르거나 뺐는지를 반환한다.
- 'esc' 키를 눌렀다면, self.close()를 통해 위젯이 종료된다.
- 'F' 키 또는 'N' 키를 눌렀다면, 위젯의 크기가 최대화되거나 보통 크기가 된다.



## Section 06 시그널과 슬롯(Signal & Slot)

### □이벤트 핸들러 재구성하기2

- 이번에는 `mouseMoveEvent`를 이용해서 마우스의 위치를 트래킹해서 출력해보자.
- `ex_6_4.py`

```
1  import sys
2
3  from PyQt6.QtWidgets import QApplication, QWidget, QLabel
4
5  class MyApp(QWidget):
6      def __init__(self):
7          super().__init__()
8          self.initUI()
9
10     def initUI(self):
11         x = 0
12         y = 0
```

# Section 06 시그널과 슬롯(Signal & Slot)

## □이벤트 핸들러 재구성하기2

- ex\_6\_4.py

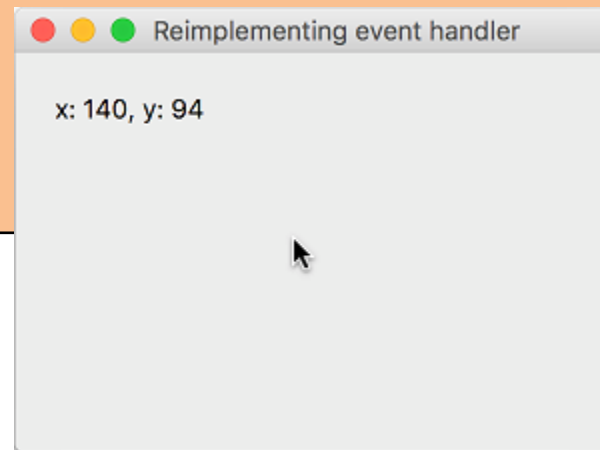
```
13         self.text = 'x: {0}, y: {1}'.format(x, y)
14         self.label = QLabel(self.text, self)
15         self.label.move(20, 20)
16
17         self.setMouseTracking(True) # Enable mouse tracking
18
19         self.setWindowTitle('Reimplementing event handler')
20         self.setGeometry(300, 300, 300, 200)
21         self.show()
22
23     def mouseMoveEvent(self, e):
24         x = e.position().x() # Use position() method to get QPoint object in PyQt6
25         y = e.position().y()
26
```

# Section 06 시그널과 슬롯(Signal & Slot)

## □ 이벤트 핸들러 재구성하기2

- ex\_6\_4.py

```
27     text = 'x: {0}, y: {1}'.format(x, y)
28     self.label.setText(text)
29     self.label.adjustSize()
30
31 if __name__ == '__main__':
32     app = QApplication(sys.argv)
33     ex = MyApp()
34     app.exec()
```



## Section 06 시그널과 슬롯(Signal & Slot)

### □이벤트 핸들러 재구성하기2

```
self.text = 'x: {0}, y: {1}'.format(x, y)
self.label = QLabel(self.text, self)
self.label.move(20, 20)
```

- x, y의 값을 self.text로 저장하고, self.label의 텍스트로 설정한다.
- 위치를 x=20, y=20 만큼 이동시킨다.

```
self.setMouseTracking(True)
```

- setMouseTracking을 True로 설정해주면, 마우스의 위치를 트래킹한다.
- 디폴트는 setMouseTracking(False) 상태이며, 마우스 버튼을 클릭하거나 뿔 때만 mouseEvent가 발생한다.

## Section 06 시그널과 슬롯(Signal & Slot)

### □이벤트 핸들러 재구성하기2

```
def mouseMoveEvent(self, e):  
  
    x = e.x()  
    y = e.y()  
  
    text = 'x: {0}, y: {1}'.format(x, y)  
    self.label.setText(text)  
    self.label.adjustSize()
```

- 이벤트 `e`는 이벤트에 대한 정보를 갖고 있는 하나의 객체이다.
- 이 이벤트 객체 (event object)는 생성된 이벤트의 유형에 따라 다르다.
- `e.x()`, `e.y()`는 위젯 안에서 이벤트가 발생했을 때 마우스 커서의 위치를 반환한다.
- 만약 `e.globalX()`, `e.globalY()`로 설정해주면, 화면 전체에서 마우스 커서의 위치를 반환하게 된다.

## Section 06 시그널과 슬롯(Signal & Slot)

### □사용자 정의 시그널

- 지정되어 있는 시그널 말고, 새로 원하는 시그널을 만들어서 사용할 수도 있다.
- 이번 예제에서는 `pyqtSignal()`을 이용해서 사용자 정의 시그널을 만들고, 특정 이벤트가 발생했을 때 이 시그널이 방출되도록 해보자.
- `ex_6_5.py`

```
1  import sys
2  from PyQt6.QtCore import pyqtSignal, QObject
3  from PyQt6.QtWidgets import QApplication, QMainWindow
4
5  class Communicate(QObject):
6      closeApp = pyqtSignal()
7
8  class MyApp(QMainWindow):
9      def __init__(self):
10         super().__init__()
```

## Section 06 시그널과 슬롯(Signal & Slot)

### □사용자 정의 시그널

- ex\_6\_5.py

```
11         self.initUI()
12
13     def initUI(self):
14         self.c = Communicate()
15         self.c.closeApp.connect(self.close)
16
17         self.setWindowTitle('Emitting Signal')
18         self.setGeometry(300, 300, 300, 200)
19         self.show()
20
21     def mousePressEvent(self, e):
22         self.c.closeApp.emit()
23
```

## Section 06 시그널과 슬롯(Signal & Slot)

### □사용자 정의 시그널

- ex\_6\_5.py

```
24  if __name__ == '__main__':  
25      app = QApplication(sys.argv)  
26      ex = MyApp()  
27      app.exec()
```

- 이 예제는 closeApp이라는 시그널을 하나 만들었다.
- 이 시그널은 마우스 클릭 시 발생해서 QMainWindow의 close() 슬롯에 연결되어 프로그램을 종료한다.



## Section 06 시그널과 슬롯(Signal & Slot)

### □사용자 정의 시그널

```
class Communicate(QObject):  
  
    closeApp = pyqtSignal()
```

- **pyqtSignal()**을 가지고 **Communicate** 클래스의 속성으로서 **closeApp**이라는 시그널을 하나 만들었다.

```
self.c = Communicate()  
self.c.closeApp.connect(self.close)
```

- **Communicate** 클래스의 **closeApp** 시그널은 **MyApp** 클래스의 **close()** 슬롯에 연결된다.

## Section 06 시그널과 슬롯(Signal & Slot)

---

### □사용자 정의 시그널

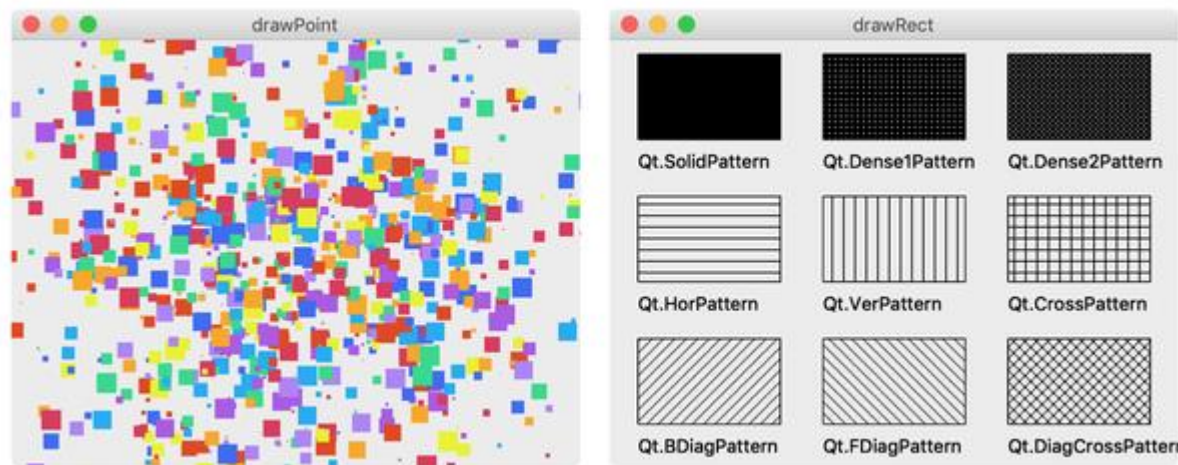
```
def mousePressEvent(self, e):  
  
    self.c.closeApp.emit()
```

- **mousePressEvent** 이벤트 핸들러를 사용해서, 마우스를 클릭했을 때 **closeApp** 시그널이 방출되도록 했다.

## Section 07 그림 그리기 (Painting)

### □개요

- PyQt5.QtGui의 QPainter 클래스는 GUI 프로그램에서 필요한 ‘그리기’와 관련한 기능들을 제공한다.
- 간단한 점, 선에서 다각형, 타원과 같은 여러 도형, 텍스트까지 다양한 방식으로 그릴 수 있다.
- 예제를 통해 PyQt5의 다양한 그림 그리기 작업에 대해 다뤄 보자.



## Section 07 그림 그리기 (Painting)

### □점 그리기 (drawPoint)

- QPainter의 동작은 일반적으로 위젯의 페인트 이벤트 (paint event)를 통해 이루어진다.
- 아래의 예제에서는 drawPoint() 메서드를 이용해서 위젯에 다양한 방식으로 점을 그려 보자.
- ex\_7\_1\_1.py

```
1  import sys
2  from PyQt6.QtWidgets import QWidget, QApplication
3  from PyQt6.QtGui import QPainter, QPen
4  from PyQt6.QtCore import Qt
5
6  class MyApp(QWidget):
7      def __init__(self):
8          super().__init__()
9          self.initUI()
10
```

# Section 07 그림 그리기 (Painting)

## □점 그리기 (drawPoint)

- ex\_7\_1\_1.py

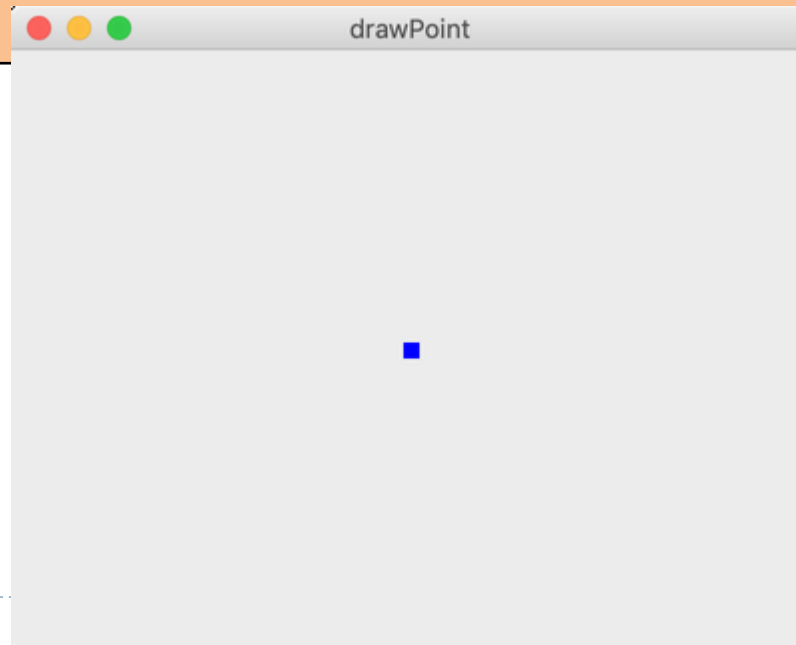
```
11     def initUI(self):
12         self.setGeometry(300, 300, 400, 300)
13         self.setWindowTitle('Points')
14         self.show()
15
16     def paintEvent(self, e):
17         qp = QPainter()
18         qp.begin(self)
19         self.draw_point(qp)
20         qp.end()
21
22     def draw_point(self, qp):
23         qp.setPen(QPen(Qt.GlobalColor.blue, 8)) # Adjusted for PyQt6
24         qp.drawPoint(int(self.width()/2), int(self.height()/2)) # Ensure coordinates are integers
```

## Section 07 그림 그리기 (Painting)

### □점 그리기 (drawPoint)

- ex\_7\_1\_1.py

```
25  
26 if __name__ == '__main__':  
27     app = QApplication(sys.argv)  
28     ex = MyApp()  
29     app.exec()
```



## Section 07 그림 그리기 (Painting)

### □점 그리기 (drawPoint)

- setPen()을 이용해서 점의 색깔과 크기를 조절하면서 세 개의 점을 그렸다.

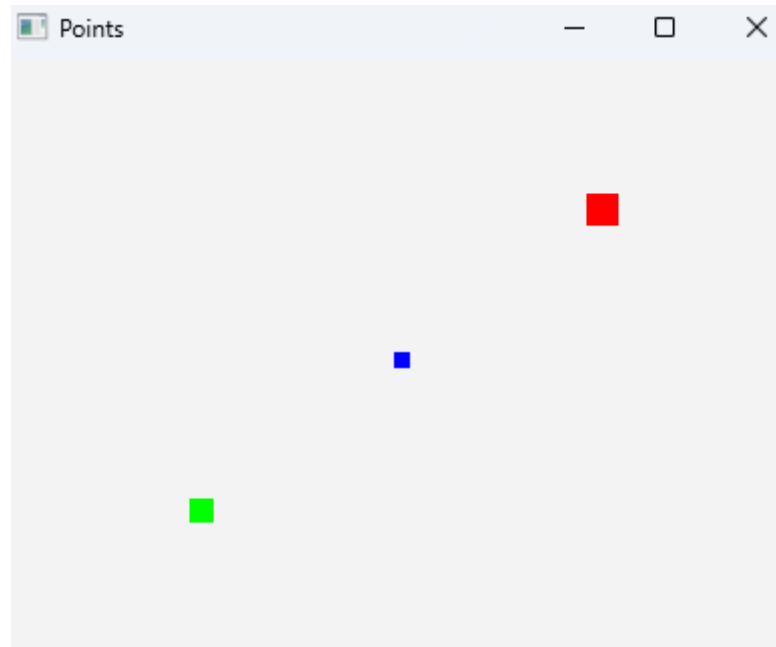
```
22     def draw_point(self, qp):
23         # Set pen with blue color and draw a point
24         qp.setPen(QPen(Qt.GlobalColor.blue, 8))
25         qp.drawPoint(int(self.width() / 2), int(self.height() / 2))
26
27         # Set pen with green color and draw a point
28         qp.setPen(QPen(Qt.GlobalColor.green, 12))
29         qp.drawPoint(int(self.width() / 4), int(3 * self.height() / 4))
30
31         # Set pen with red color and draw a point
32         qp.setPen(QPen(Qt.GlobalColor.red, 16))
33         qp.drawPoint(int(3 * self.width() / 4), int(self.height() / 4))
```

## Section 07 그림 그리기 (Painting)

---

### □점 그리기 (drawPoint)

- 결과는 아래와 같다.





# Section 07 그림 그리기 (Painting)

## □점 그리기 (drawPoint)

- ex\_7\_1\_2.py

```
1 import sys
2 import numpy as np
3 from PyQt6.QtWidgets import QWidget, QApplication
4 from PyQt6.QtGui import QPainter, QPen, QColor
5
6 class MyApp(QWidget):
7     def __init__(self):
8         super().__init__()
9         self.initUI()
10
11     def initUI(self):
12         self.setGeometry(300, 300, 400, 300)
13         self.setWindowTitle('drawPoint')
14         self.show()
15
```

# Section 07 그림 그리기 (Painting)

## □점 그리기 (drawPoint)

- ex\_7\_1\_2.py

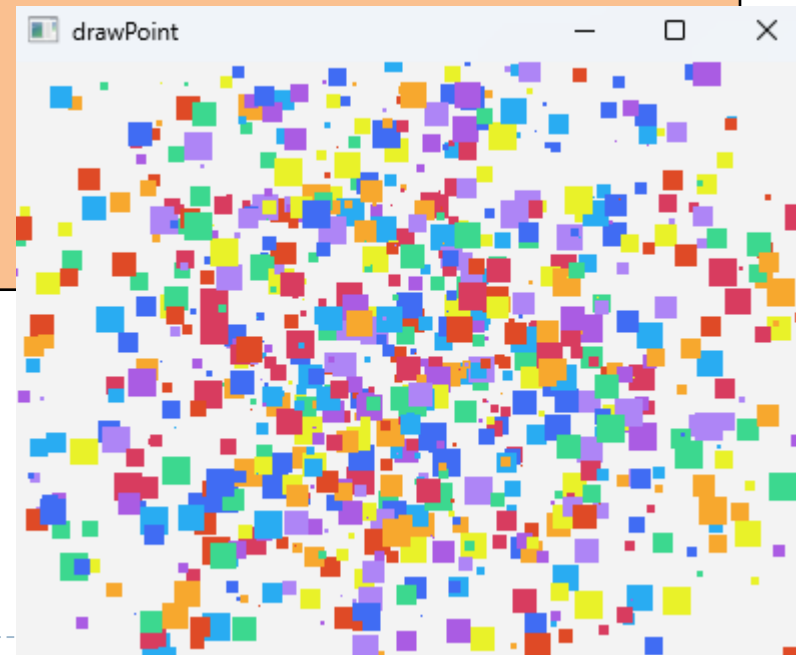
```
16     def paintEvent(self, e):
17         qp = QPainter(self) # Initiate QPainter with the widget
18         self.draw_point(qp)
19         qp.end()
20
21     def draw_point(self, qp):
22         pen = QPen()
23         colors = ['#D83C5F', '#3CD88F', '#AA5CE3',
24                  '#DF4A26', '#AE85F6', '#F7A82E',
25                  '#406CF3', '#E9F229', '#29ACF2']
26
27         for i in range(1000):
28             pen.setWidth(np.random.randint(1, 15))
29             pen.setColor(QColor(np.random.choice(colors)))
30             qp.setPen(pen)
```

## Section 07 그림 그리기 (Painting)

### □점 그리기 (drawPoint)

- ex\_7\_1\_2.py

```
29     rand_x = int(100 * np.random.randn()) # Ensure the value is an integer
30     rand_y = int(100 * np.random.randn()) # Ensure the value is an integer
31     qp.drawPoint(int(self.width() / 2 + rand_x), int(self.height() / 2 + rand_y))
32
33 if __name__ == '__main__':
34     app = QApplication(sys.argv)
35     ex = MyApp()
36     app.exec()
```



## Section 07 그림 그리기 (Painting)

---

### □점 그리기 (drawPoint)

- `setWidth()`와 `setColor()`는 각각 펜 (`QPen()`)의 선의 너비와 색깔을 설정한다.
- `numpy.random` 모듈을 이용해서 점의 크기와 색깔을 임의로 선택하도록 하고 `drawPoint()`를 이용해서 점을 그린다.

## Section 07 그림 그리기 (Painting)

### □ 직선 그리기 (drawLine)

- 이번 예제에서는 drawLine()을 이용해서 위젯에 다양한 스타일의 직선을 그려 보자.
- ex\_7\_2.py

```
1  import sys
2  from PyQt6.QtWidgets import QWidget, QApplication
3  from PyQt6.QtGui import QPainter, QColor, QPen
4  from PyQt6.QtCore import Qt
5
6  class MyApp(QWidget):
7      def __init__(self):
8          super().__init__()
9          self.initUI()
10
11     def initUI(self):
12         self.setGeometry(300, 300, 400, 300)
```

# Section 07 그림 그리기 (Painting)

## □ 직선 그리기 (drawLine)

- ex\_7\_2.py

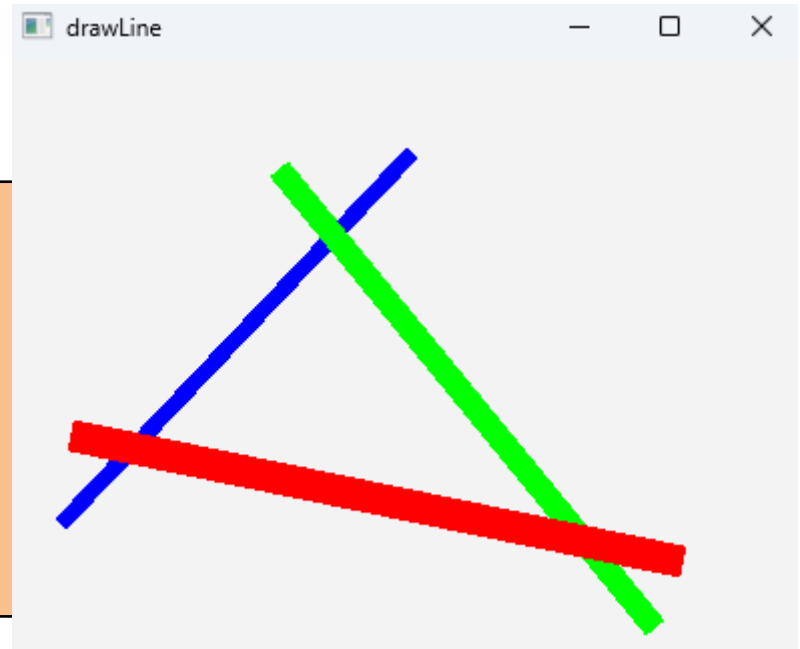
```
13         self.setWindowTitle('drawLine')
14         self.show()
15
16     def paintEvent(self, e):
17         qp = QPainter(self) # Direct initialization with the widget
18         self.draw_line(qp)
19         qp.end()
20
21     def draw_line(self, qp):
22         qp.setPen(QPen(Qt.GlobalColor.blue, 8))
23         qp.drawLine(30, 230, 200, 50)
24         qp.setPen(QPen(Qt.GlobalColor.green, 12))
25         qp.drawLine(140, 60, 320, 280)
26         qp.setPen(QPen(Qt.GlobalColor.red, 16))
```

## Section 07 그림 그리기 (Painting)

### □ 직선 그리기 (drawLine)

- ex\_7\_2.py

```
27         qp.drawLine(330, 250, 40, 190)
28
29     if __name__ == '__main__':
30         app = QApplication(sys.argv)
31         ex = MyApp()
32         app.exec()
```



- 세 개의 직선이 그려졌다.
- `setPen()`을 이용해서 선의 색깔과 두께를 설정해주고, `drawLine(x1, y1, x2, y2)`의 형태로 선의 양 끝점의 위치를 정수로 입력한다.

# Section 07 그림 그리기 (Painting)

## □ 직선 그리기 (drawLine)

- ex\_7\_2.py

```
21     def draw_line(self, qp):
22         qp.setPen(QPen(QColor(Qt.GlobalColor.black), 3, Qt.PenStyle.SolidLine))
23         qp.drawLine(20, 20, 380, 20)
24         qp.drawText(30, 40, 'Qt.SolidLine')
25
26         qp.setPen(QPen(QColor(Qt.GlobalColor.black), 3, Qt.PenStyle.DashLine))
27         qp.drawLine(20, 70, 380, 70)
28         qp.drawText(30, 90, 'Qt.DashLine')
29
30         qp.setPen(QPen(QColor(Qt.GlobalColor.black), 3, Qt.PenStyle.DotLine))
31         qp.drawLine(20, 120, 380, 120)
32         qp.drawText(30, 140, 'Qt.DotLine')
33
34         qp.setPen(QPen(QColor(Qt.GlobalColor.black), 3, Qt.PenStyle.DashDotLine))
```



## Section 07 그림 그리기 (Painting)

### □직선 그리기 (drawLine)

- ex\_7\_2.py

```
35     qp.drawLine(20, 170, 380, 170)
36     qp.drawText(30, 190, 'Qt.DashDotLine')
37
38     qp.setPen(QPen(QColor(Qt.GlobalColor.black), 3, Qt.PenStyle.DashDotDotLine))
39     qp.drawLine(20, 220, 380, 220)
40     qp.drawText(30, 240, 'Qt.DashDotDotLine')
41
42     pen = QPen(QColor(Qt.GlobalColor.black), 3, Qt.PenStyle.CustomDashLine)
43     pen.setDashPattern([1, 2, 3, 4]) # Example pattern, adjust as needed
44     qp.setPen(pen)
45     qp.drawLine(20, 270, 380, 270)
46     qp.drawText(30, 290, 'Qt.CustomDashLine')
47
```

## Section 07 그림 그리기 (Painting)

### □직선 그리기 (drawLine)

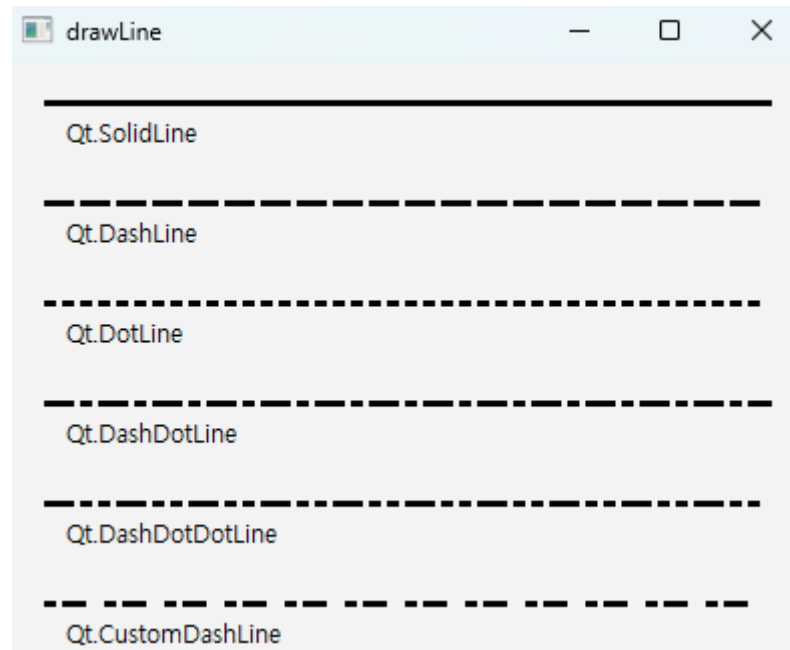
- ex\_7\_2.py

```
35     qp.drawLine(20, 170, 380, 170)
36     qp.drawText(30, 190, 'Qt.DashDotLine')
37
38     qp.setPen(QPen(QColor(Qt.GlobalColor.black), 3, Qt.PenStyle.DashDotDotLine))
39     qp.drawLine(20, 220, 380, 220)
40     qp.drawText(30, 240, 'Qt.DashDotDotLine')
41
42     pen = QPen(QColor(Qt.GlobalColor.black), 3, Qt.PenStyle.CustomDashLine)
43     pen.setDashPattern([1, 2, 3, 4]) # Example pattern, adjust as needed
44     qp.setPen(pen)
45     qp.drawLine(20, 270, 380, 270)
46     qp.drawText(30, 290, 'Qt.CustomDashLine')
47
```

## Section 07 그림 그리기 (Painting)

### □ 직선 그리기 (drawLine)

- ex\_7\_2.py
- Qt.SolidLine, Qt.DashLine 등을 입력하면 다양한 형태의 선 종류를 설정할 수 있다.
- 결과는 아래 그림과 같다.



## Section 07 그림 그리기 (Painting)

### □ 직사각형 그리기 (drawRect)

- drawRect() 메서드를 이용해서 다양한 스타일의 직사각형을 그릴 수 있다.
- drawRect()에 x, y, width, height 순서로 숫자를 입력한다.
- ex\_7\_3.py

```
1  import sys
2
3  from PyQt6.QtWidgets import QWidget, QApplication
4
5  from PyQt6.QtGui import QPainter, QColor, QPen, QBrush
6
7  from PyQt6.QtCore import Qt
8
9
10
11 class MyApp(QWidget):
12     def __init__(self):
13         super().__init__()
14         self.initUI()
15
16     def initUI(self):
17         self.setGeometry(300, 300, 400, 300)
```

# Section 07 그림 그리기 (Painting)

## □직사각형 그리기 (drawRect)

- ex\_7\_3.py

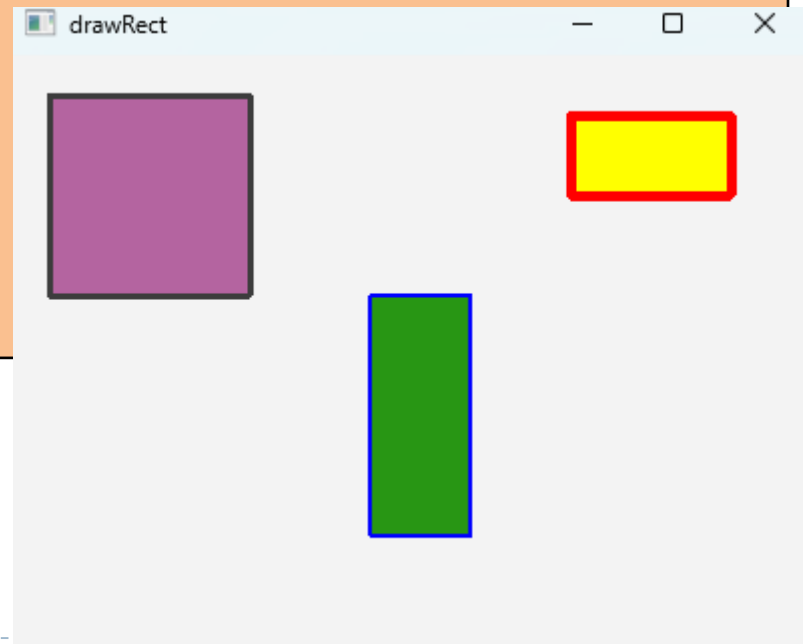
```
13         self.setWindowTitle('drawRect')
14         self.show()
15
16     def paintEvent(self, e):
17         qp = QPainter(self) # QPainter is now initialized with the widget
18         self.draw_rect(qp)
19         qp.end()
20
21     def draw_rect(self, qp):
22         qp.setBrush(QColor(180, 100, 160))
23         qp.setPen(QPen(QColor(60, 60, 60), 3))
24         qp.drawRect(20, 20, 100, 100)
25         qp.setBrush(QColor(40, 150, 20))
26         qp.setPen(QPen(QColor.fromRgb(0, 0, 255), 2)) # Explicitly use QColor for blue
27         qp.drawRect(180, 120, 50, 120)
```

## Section 07 그림 그리기 (Painting)

### □ 직사각형 그리기 (drawRect)

- ex\_7\_3.py

```
28
29     qp.setBrush(QColor.fromRgb(255, 255, 0)) # Explicitly use QColor for yellow
30     qp.setPen(QPen(QColor.fromRgb(255, 0, 0), 5)) # Explicitly use QColor for red
31     qp.drawRect(280, 30, 80, 40)
32
33 if __name__ == '__main__':
34     app = QApplication(sys.argv)
35     ex = MyApp()
36     sys.exit(app.exec())
```



# Section 07 그림 그리기 (Painting)

## □직사각형 그리기 (drawRect)

- ex\_7\_3.py

```
21     def draw_rect(self, qp):
22         brush = QBrush(Qt.BrushStyle.SolidPattern)
23         qp.setBrush(brush)
24         qp.drawRect(20, 10, 100, 60)
25         qp.drawText(20, 90, 'Qt.SolidPattern')
26
27         brush = QBrush(Qt.BrushStyle.Dense1Pattern)
28         qp.setBrush(brush)
29         qp.drawRect(150, 10, 100, 60)
30         qp.drawText(150, 90, 'Qt.Dense1Pattern')
31         brush = QBrush(Qt.BrushStyle.Dense2Pattern)
32         qp.setBrush(brush)
33         qp.drawRect(280, 10, 100, 60)
34         qp.drawText(280, 90, 'Qt.Dense2Pattern')
```

# Section 07 그림 그리기 (Painting)

## □ 직사각형 그리기 (drawRect)

- ex\_7\_3.py

```
36     brush = QBrush(Qt.BrushStyle.HorPattern)
37     qp.setBrush(brush)
38     qp.drawRect(20, 110, 100, 60)
39     qp.drawText(20, 190, 'Qt.HorPattern')
40
41     brush = QBrush(Qt.BrushStyle.VerPattern)
42     qp.setBrush(brush)
43     qp.drawRect(150, 110, 100, 60)
44     qp.drawText(150, 190, 'Qt.VerPattern')
45
46     brush = QBrush(Qt.BrushStyle.CrossPattern)
47     qp.setBrush(brush)
48     qp.drawRect(280, 110, 100, 60)
49     qp.drawText(280, 190, 'Qt.CrossPattern')
50
```



# Section 07 그림 그리기 (Painting)

## □직사각형 그리기 (drawRect)

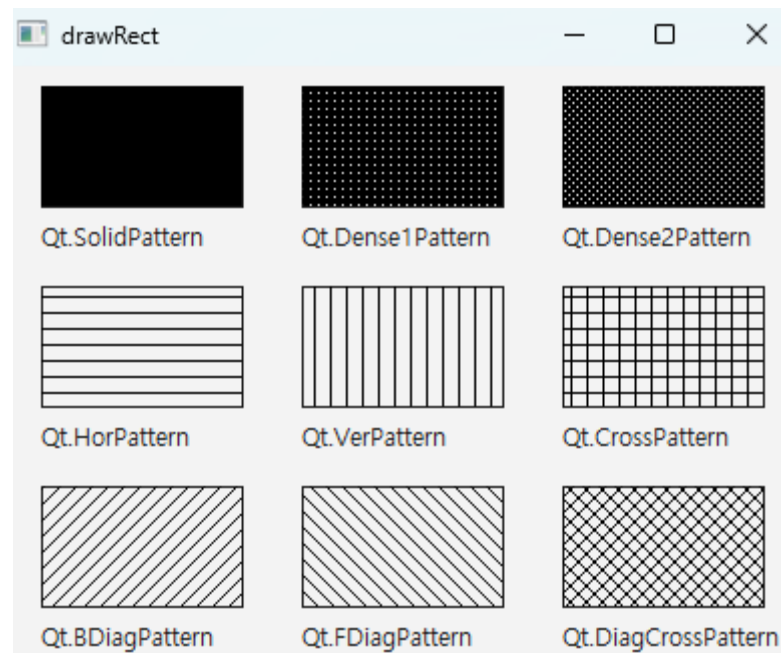
- ex\_7\_3.py

```
51     brush = QBrush(Qt.BrushStyle.BDiagPattern)
52     qp.setBrush(brush)
53     qp.drawRect(20, 210, 100, 60)
54     qp.drawText(20, 290, 'Qt.BDiagPattern')
55
56     brush = QBrush(Qt.BrushStyle.FDiagPattern)
57     qp.setBrush(brush)
58     qp.drawRect(150, 210, 100, 60)
59     qp.drawText(150, 290, 'Qt.FDiagPattern')
60
61     brush = QBrush(Qt.BrushStyle.DiagCrossPattern)
62     qp.setBrush(brush)
63     qp.drawRect(280, 210, 100, 60)
64     qp.drawText(280, 290, 'Qt.DiagCrossPattern')
65
```

## Section 07 그림 그리기 (Painting)

### □ 직사각형 그리기 (drawRect)

- Qt.SolidPattern, Qt.Dense1Pattern 등을 이용해서 직사각형의 채우기 패턴을 설정할 수 있다.
- 결과는 아래와 같다.



---

# Q&A

