

2. 레이아웃

- 레이아웃(Layout)은 어플리케이션 창에 위젯들을 배치하는 방식임
- PyQt5의 위젯들을 배치하는 방식에는 절대적 배치, 박스 레이아웃, 그리드 레이아웃 방식이 있음

(1) 절대적 배치

- 절대적 배치(Absolute positioning) 방식은 각 위젯의 위치와 크기를 픽셀 단위로 설정해서 배치함
- 창의 크기를 조절해도 위젯의 크기와 위치는 변하지 않음
- 다양한 플랫폼에서 어플리케이션이 다르게 보일 수 있음
- 어플리케이션의 폰트를 바꾸면 레이아웃이 망가질 수 있음
- 레이아웃을 바꾸고 싶다면 완전히 새로 고쳐야 함

```
#절대적 배치

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QPushButton
class MyApp(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        label1=QLabel('Label1', self)
        label1.move(20, 20)
        label2=QLabel('Label2', self)
        label2.move(20, 60)

        btn1=QPushButton('Button1', self)
        btn1.move(80, 13)
        btn2=QPushButton('Button2', self)
        btn2.move(80, 53)

        self.setWindowTitle('Absolute Positioning')
        self.setGeometry(300, 300, 400, 200)
        self.show()

if __name__ == '__main__':
    app=QApplication(sys.argv)
    ex=MyApp()
    sys.exit(app.exec_())
```

- 라벨을 하나 만들고, x=20, y=20에 위치하도록 옮김
- 푸시버튼을 하나 만들고, x=80, y=13에 위치하도록 옮김
- 위젯의 위치를 설정하기 위해 move()를 사용함
- 라벨(label1, 2)과 푸시버튼(btn1, 2)의 x, y 좌표를 설정함으로써 위치를 조절함
- x 좌표는 왼쪽에서 오른쪽으로 갈수록 커지고, y 좌표는 위에서 아래로 갈수록 커짐

(2) 박스 레이아웃

- QHBoxLayout, QVBoxLayout은 여러 위젯을 수평으로 정렬하는 레이아웃 클래스임
- QHBoxLayout, QVBoxLayout 생성자는 수평, 수직의 박스를 하나 만드는데, 다른 레이아웃 박스를 넣을 수도 있고 위젯을 배치할 수도 있음

```
#박스 레이아웃

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QHBoxLayout, QVBoxLayout
class MyApp(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        okButton=QPushButton('OK')
        cancelButton=QPushButton('Cancel')

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

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

        self.setLayout(vbox)

        self.setWindowTitle('Box Layout')
        self.setGeometry(300, 300, 300, 200)
        self.show()

if __name__ == '__main__':
    app=QApplication(sys.argv)
    ex=MyApp()
    sys.exit(app.exec_())
```

- OK, Cancel 두 개의 버튼을 만들
- 수평 박스(hbox)를 하나 만들고, 두 개의 버튼과 양쪽에 빈 공간을 추가함
- addStretch()는 신축성 있는 빈 공간을 제공함
- hbox는 두 버튼 양쪽의 stretch factor가 1로 같기 때문에 빈 공간의 크기는 창의 크기가 변화해도 항상 같음
- 수평 박스를 수직 박스(vbox)에 넣어줌, 수직 박스의 stretch factor은 수평 박스를 아래쪽으로 밀어내서 두 개의 버튼을 창의 아래쪽에 위치하도록 함
- 수평 박스 위와 아래 빈 공간의 크기는 항상 3:1을 유지함
- self.setLayout(vbox)를 통해 최종적으로 수직 박스를 창의 메인 레이아웃으로 설정함

(3) 그리드 레이아웃

- 가장 일반적인 레이아웃 클래스
- 그리드 레이아웃 클래스는 위젯의 공간을 행(row)와 열(column)로 구분함
- 레이아웃을 생성하기 위해 QGridLayout 클래스를 사용함

```
#그리드 레이아웃

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QGridLayout, QLabel,
QLineEdit, QTextEdit
class MyApp(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        grid=QGridLayout()
        self.setLayout(grid)

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

        grid.addWidget(QLineEdit(), 0, 1)
        grid.addWidget(QLineEdit(), 1, 1)
        grid.addWidget(QLineEdit(), 2, 1)

        self.setWindowTitle('QGridLayout')
        self.setGeometry(300, 300, 300, 200)
        self.show()

if __name__ == '__main__':
    app=QApplication(sys.argv)
    ex=MyApp()
    sys.exit(app.exec_())
```

- QGridLayout을 만들고, 어플리케이션 창의 레이아웃으로 설정함
- addWidget()의 첫번째 파라미터는 추가할 위젯, 두세번째 파라미터는 각각 행과 열 번호를 입력함
- QTextEdit() 위젯은 QLineEdit() 위젯과 달리 여러 줄의 텍스트를 수정할 수 있는 위젯임

3. 위젯(1)

- 위젯은 어플리케이션을 이루는 기본적인 구성 요소

(1) QPushButton

- 푸시 버튼 또는 명령 버튼은 사용자의 명령에 따라 프로그램이 어떤 동작을 하도록 할 때 사용되는 버튼
- 자주 쓰이는 메소드

메소드	설명
setCheckable()	True 설정 시, 누른 상태와 그렇지 않은 상태를 구분함
toggle()	상태를 바꿈
setIcon()	버튼의 아이콘을 설정함
setEnabled()	False 설정 시, 버튼을 사용할 수 없음
isChecked()	버튼의 선택 여부를 반환함
setText()	버튼에 표시될 텍스트를 설정함
text()	버튼에 표시된 텍스트를 반환함

- 자주 쓰이는 시그널

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

```
#QPushButton
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QVBoxLayout
class MyApp(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        btn1=QPushButton('&Button1', self)
        btn1.setCheckable(True)
        btn1.toggle()

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

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

        vbox=QVBoxLayout()
        vbox.addWidget(btn1)
        vbox.addWidget(btn2)
        vbox.addWidget(btn3)

        self.setLayout(vbox)
        self.setWindowTitle('QPushButton')
        self.setGeometry(300, 300, 300, 200)
```

```
self.show()

if __name__ == '__main__':
    app=QApplication(sys.argv)
    ex=MyApp()
    sys.exit(app.exec_())
```

- QPushButton 클래스로 푸시 버튼을 하나 만듦. 첫번째 파라미터로는 버튼에 표시될 텍스트, 두번째는 버튼이 속할 부모 클래스를 지정함
- setCheckable()을 True로 설정해주면, 선택되거나 선택되지 않은 상태를 유지할 수 있음
- toggle()을 호출하면 버튼의 상태가 바뀌게 됨. 따라서 이 버튼은 프로그램이 시작될 때 선택되어 있음.
- setText()를 사용해서 버튼에 표시될 텍스트를 지정함
- setEnabled()를 False로 설정하면 버튼을 사용할 수 없음

(2) QLabel

- QLabel 위젯은 텍스트 또는 이미지 라벨을 만들 때 쓰임
- 사용자와 어떤 상호작용을 제공하진 않음

```
#QLabel

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QVBoxLayout
from PyQt5.QtCore import Qt
class MyApp(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        label1=QLabel('First Label', self)
        label1.setAlignment(Qt.AlignCenter)

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

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

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

        label1.setFont(font1)
        label2.setFont(font2)

        layout=QVBoxLayout()
        layout.addWidget(label1)
        layout.addWidget(label2)

        self.setLayout(layout)

        self.setWindowTitle('QLabel')
        self.setGeometry(300, 300, 300, 200)
        self.show()

if __name__ == '__main__':
    app=QApplication(sys.argv)
    ex=MyApp()
    sys.exit(app.exec_())
```

- QLabel 생성자에 라벨 텍스트와 부모 위젯을 입력해줌
- setAlignment()로 라벨의 배치를 설정할 수 있음
- Qt.AlignCenter로 설정해주면 수평, 수직 방향 모두 가운데 위치함
- setPointSize()로 폰트의 크기를 설정해줌
- Qt.AlignVCenter은 수직 방향으로 가운데를 말함
- setFamily()는 폰트의 종류를 설정해줌
- setBold를 True로 설정하면 폰트를 진하게 만들어줌, 폰트 크기를 설정하지 않으면 디폴트 크기인 13으로 자동 설정됨

(3) QCheckBox

- QCheckBox 위젯은 on/off 두 상태를 갖는 버튼을 제공함
- 체크 박스가 선택되거나 해제될 때, stateChanged()을 발생함
- 체크 박스의 선택 여부를 확인하기 위해, isChecked()를 사용할 수 있으며, 선택 여부에 따라 boolean 값을 반환함
- setTristate()를 사용하면 '변경 없음' 상태를 가질 수 있으며, 사용자에게 선택하거나 선택하지 않을 옵션을 줄 때 유용함
- 세 가지 상태를 갖는 체크 박스의 상태를 얻기 위해서는 checkState()를 사용하며, 선택/변경 없음/해제 여부에 따라 각각 2/1/0의 값을 반환함
- QPushButton 클래스를 사용하면 여러 개의 버튼을 묶어 exclusive/non-exclusive 버튼 그룹을 만들 수 있으며, exclusive 버튼 그룹은 여러 개 중 하나의 버튼만 선택 가능함
- 자주 쓰이는 메소드

메소드	설명
text()	체크 박스의 라벨 텍스트 반환
setText()	체크 박스의 라벨 텍스트 설정
isChecked()	체크 박스의 상태 반환(True/False)
checkState()	체크 박스의 상태 반환(2/1/0)
toggle()	체크 박스의 상태 변경

- 자주 쓰이는 시그널

시그널	설명
pressed()	체크 박스를 누를 때 신호 발생
released()	체크 박스에서 떼를 때 신호 발생
clicked()	체크 박스를 클릭할 때 신호 발생
stateChanged()	체크 박스의 상태가 바뀔 때 신호 발생

```
#QCheckBox

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QCheckBox
from PyQt5.QtCore import Qt
class MyApp(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        cb=QCheckBox('Show title', self)
        cb.move(20, 20)
        cb.toggle()
        cb.stateChanged.connect(self.changeTitle)

        self.setWindowTitle('QCheckBox')
        self.setGeometry(300, 300, 300, 200)
        self.show()

    def changeTitle(self, state):
        if state==Qt.Checked:
            self.setWindowTitle('QCheckBox')
        else:
            self.setWindowTitle(' ')
if __name__ == '__main__':
```

```
app=QApplication(sys.argv)
ex=MyApp()
sys.exit(app.exec_())
```

- 'Show title'이라는 텍스트 라벨을 갖는 체크박스를 하나 만듦
- 체크 박스는 디폴트로 off 상태로 나타나기 때문에 on 상태로 바꾸기 위해 toggle() 사용
- 체크박스의 상태가 바뀔 때 발생하는 시그널 (stateChanged)을 우리가 정의한 changeTitle()에 연결함
- 체크박스의 상태가 changeTitle()의 매개변수로 주어짐
- 체크가 되어있으면 타이틀을 'QCheckBox'로, 그렇지 않으면 빈 문자열로 나타내게 됨

(4) QRadioButton

- QRadioButton 위젯은 사용자에게 여러 개 중 하나의 옵션을 선택하도록 할 때 사용됨
- 한 위젯 안에 여러 라디오 버튼은 기본적으로 autoExclusive로 설정되어 있으며, 하나의 버튼을 선택하면 나머지 버튼들은 선택 해제가 됨
- 한 번에 여러 버튼을 선택할 수 있도록 하려면 setAutoExclusive()에 False를 입력함
- 한 위젯 안에 여러 개의 exclusive 버튼 그룹을 배치하고 싶으면 QButtonGroup()을 사용
- 자주 쓰이는 메소드

메소드	설명
text()	버튼의 텍스트를 반환
setText()	라벨에 들어갈 텍스트 설정
setChecked()	버튼의 선택 여부 설정
isChecked()	버튼의 선택 여부 반환
toggle()	버튼의 상태 변경

- 자주 쓰이는 시그널

시그널	설명
pressed()	버튼을 누를 때 신호 발생
released()	버튼에서 뗄 때 신호 발생
clicked()	버튼을 클릭할 때 신호 발생
toggled()	버튼의 상태가 바뀔 때 신호 발생

```
#QRadioButton

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QRadioButton
class MyApp(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        rbtn1=QRadioButton('First Button', self)
        rbtn1.move(50, 50)
        rbtn1.setChecked(True)

        rbtn2=QRadioButton(self)
        rbtn2.move(50, 70)
        rbtn2.setText('Second Button')

        self.setGeometry(300, 300, 300, 200)
        self.setWindowTitle('QRadioButton')
        self.show()

if __name__ == '__main__':
    app=QApplication(sys.argv)
    ex=MyApp()
    sys.exit(app.exec_())
```

- QRadioButton을 이용해 라디오 버튼을 하나 만들고, 라벨에 들어갈 텍스트와 부모 위젯을 입력함
- setChecked()를 True로 설정하면 프로그램이 실행될 때 버튼이 선택되어 표시됨

(5) QComboBox

- QComboBox는 작은 공간을 차지하면서, 여러 옵션 중 하나의 옵션을 선택하도록 함

```
#QComboBox

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QComboBox
class MyApp(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.lbl=QLabel('Option1', self)
        self.lbl.move(50, 150)

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

        cb.activated[str].connect(self.onActivated)

        self.setWindowTitle('QComboBox')
        self.setGeometry(300, 300, 300, 200)
        self.show()

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

if __name__ == '__main__':
    app=QApplication(sys.argv)
    ex=MyApp()
    sys.exit(app.exec_())
```

- QComboBox 위젯을 하나 만들고, addItem()을 이용해 선택 가능한 4개의 옵션을 추가함
- 옵션을 선택하면, onActivated()가 호출됨
- 선택한 항목의 텍스트가 라벨에 나타나도록 하고, adjustSize()를 이용해 라벨의 크기를 자동 조절함

(6) QLineEdit

- QLineEdit은 한 줄의 문자열을 입력하고 수정할 수 있도록 하는 위젯임
- echoMode()를 설정함으로써 '쓰기 전용' 영역으로 사용할 수 있으며, 비밀번호와 같은 입력을 받을 때 유용하게 사용됨, setEchoMode()로 설정함
- setEchoMode의 매개변수

상수	값	설명
QLineEdit.Normal	0	입력된 문자를 표시함(기본값)
QLineEdit.NoEcho	1	문자열을 표시하지 않음
QLineEdit.Password	2	입력된 문자 대신 비밀번호 가림용 문자 표시
QLineEdit.PasswordEchoOnEdit	3	입력만 문자 표시, 수정 중에는 다른 문자 표시

- maxLength()로 입력되는 텍스트의 길이를 제한함
- setValidator()로 입력되는 텍스트의 종류를 제한함
- setText() 또는 insert()로 텍스트를 편집할 수 있고, text()로 입력된 텍스트를 가져옴
- echoMode에 의해 입력되는 텍스트와 표시되는 텍스트가 다르다면, displayText()로 표시되는 텍스트를 가져올 수 있음
- setSelection(), selectAll()로 텍스트를 선택하거나, cut(), copy(), paste()를 통해 잘라내기, 복사하기, 붙여넣기 등의 동작을 수행할 수 있음
- setAlignment()로 텍스트를 정렬할 수 있음
- 자주 쓰이는 시그널

시그널	설명
cursorPositionChanged()	커서가 움직일 때 발생하는 신호를 발생함
editingFinished()	편집이 끝났을 때 (Return/Enter 버튼 클릭 시) 신호 발생
selectionChanged()	선택 영역이 바뀔 때 신호 발생
textChanged()	텍스트가 변경될 때 신호 발생
textEdited()	텍스트가 편집될 때 신호 발생

```
#QLineEdit

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit
class MyApp(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.lbl=QLabel(self)
        self.lbl.move(60, 40)

        qle=QLineEdit(self)
        qle.move(60, 100)
        qle.textChanged[str].connect(self.onChanged)

        self.setWindowTitle('QLineEdit')
        self.setGeometry(300, 300, 300, 200)
        self.show()

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

```
if __name__ == '__main__':  
    app=QApplication(sys.argv)  
    ex=MyApp()  
    sys.exit(app.exec_())
```

- QLineEdit 위젯을 하나 만듦
- qle의 텍스트가 바뀌면, Onchanged()를 호출함
- onChanged() 안에서, 입력된 'text'를 라벨 위젯(lbl)의 텍스트로 설정하도록 함
- adjustSize()로 텍스트의 길이에 따라 라벨의 길이를 조절해주도록 함

(7) QProgressBar

- QProgressBar 위젯은 수평, 수직의 진행 표시줄을 제공함
- setMinimum()과 setMaximum()으로 진행 표시줄의 최소값과 최대값을 설정할 수 있으며, 또는 setRange()로 한번에 범위를 설정할 수도 있음(기본값 0, 99)
- setValue()로 진행 표시줄의 진행 상태를 특정 값으로 설정할 수 있고, reset()은 초기 상태로 되돌림
- 진행 표시줄의 최소값과 최대값을 모두 0으로 설정하면, 항상 진행 중인 상태로 표시됨



```
#QProgressBar

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QProgressBar
from PyQt5.QtCore import QBasicTimer
class MyApp(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.pbar=QProgressBar(self)
        self.pbar.setGeometry(30, 40, 200, 25)

        self.btn=QPushButton('Start', self)
        self.btn.move(40, 80)
        self.btn.clicked.connect(self.doAction)

        self.timer=QBasicTimer()
        self.step=0

        self.setWindowTitle('QProgressBar')
        self.setGeometry(300, 300, 300, 200)
        self.show()

    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)

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

if __name__ == '__main__':
    app=QApplication(sys.argv)
    ex=MyApp()
    sys.exit(app.exec_())
```

- QProgressBar 생성자로 진행 표시줄을 하나 만들, 진행 표시줄 활성화를 위해 타이머 객체 사용

- 타이머 이벤트를 실행하기 위해 `start()` 호출, 이 메소드의 첫번째 매개변수는 종료시간, 두번째 매개변수는 이벤트가 수행될 객체임
- 각각의 `QObject`와 그 자손들은 `timerEvent()` 이벤트 핸들러를 가지며, 타이머 이벤트에 반응하기 위해 이벤트 핸들러를 재구성해줌
- `doAction()` 안에서, 타이머를 시작하고 멈추도록 해줌

(8) QSlider&QDial

- QSlider은 수평 또는 수직 방향의 슬라이더를 제공하며, 슬라이더는 제한된 범위 안에서 값을 조절하는 위젯임
- 슬라이더의 틱(tick)의 간격을 조절하기 위해서는 setTickInterval(), 틱의 위치를 조절하기 위해서는 setTickPosition() 사용
- setTickInterval()의 입력값은 픽셀이 아니라 값을 의미함
- setTickPosition()의 입력값과 기능

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

- QDial은 슬라이더를 둥근 형태로 표현한 다이얼 위젯이며, 기본적으로 같은 시그널과 슬롯, 메소드들을 공유함
- 다이얼 위젯에 노치(눈금, notch)를 표시하기 위해서는 setNotchesVisible()을 사용하며, True로 설정하면 둥근 다이얼을 따라 노치들이 표시됨
- 기본적으로 노치는 표시되지 않도록 설정되어 있음
- QSlider과 QDial 위젯에서 가장 자주 쓰이는 시그널

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

```
#QSlider&QDial

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QSlider, QDial, QPushButton
from PyQt5.QtCore import Qt
class MyApp(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.slider=QSlider(Qt.Horizontal, self)
        self.slider.move(30, 30)
        self.slider.setRange(0, 50)
        self.slider.setSingleStep(2)

        self.dial=QDial(self)
        self.dial.move(30, 50)
        self.dial.setRange(0, 50)

        btn=QPushButton('Default', self)
        btn.move(35, 160)
```

```
self.slider.valueChanged.connect(self.dial.setValue)
self.dial.valueChanged.connect(self.slider.setValue)
btn.clicked.connect(self.button_clicked)
```

```
self.setWindowTitle('QSlider and QDial')
self.setGeometry(300, 300, 400, 200)
self.show()
```

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

```
if __name__ == '__main__':
    app=QApplication(sys.argv)
    ex=MyApp()
    sys.exit(app.exec_())
```

- QSlider 위젯을 하나 만들고, Qt.Horizontal 또는 Qt.Vertical을 입력해줌으로써 수평 또는 수직 방향을 설정
- setRange()로 값의 범위를 0부터 50으로 설정함, setSingleStep()은 조절 가능한 최소 단위를 설정함
- QDial 위젯을 하나 만들고, 슬라이더와 마찬가지로 setRange()로 범위를 설정함
- 슬라이더와 다이얼의 값이 변할 때 발생하는 시그널을 각각 다이얼과 슬라이더의 값을 조절해주는 메소드(setValue)에 서로 연결함으로써 두 위젯의 값이 언제나 일치하도록 함
- 'Default' 푸시 버튼을 클릭하면 발생하는 시그널을 button_clicked에 연결함
- button_clicked()은 슬라이더와 다이얼의 값을 모두 0으로 조절함, 따라서 'Default' 푸시 버튼을 클릭하면, 슬라이더와 다이얼의 값이 0으로 초기화됨

(9) QSplitter

- QSplitter 클래스는 스플리터 위젯을 구현함, 스플리터는 경계를 드래그하여 자식 위젯의 크기를 조절할 수 있도록 함

```
#QSplitter

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QHBoxLayout, QFrame,
QSplitter
from PyQt5.QtCore import Qt
class MyApp(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        hbox=QHBoxLayout()

        top=QFrame()
        top.setFrameShape(QFrame.Box)

        midleft=QFrame()
        midleft.setFrameShape(QFrame.StyledPanel)

        midright=QFrame()
        midright.setFrameShape(QFrame.Panel)

        bottom=QFrame()
        bottom.setFrameShape(QFrame.WinPanel)
        bottom.setFrameShadow(QFrame.Sunken)

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

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

        hbox.addWidget(splitter2)
        self.setLayout(hbox)

        self.setGeometry(300, 300, 300, 200)
        self.setWindowTitle('QSplitter')
        self.show()

if __name__ == '__main__':
    app=QApplication(sys.argv)
    ex=MyApp()
    sys.exit(app.exec_())
```

- 각 영역에 들어갈 프레임을 만들어주고, 프레임의 형태와 그림자의 스타일을 setFrameShape와 setFrameShadow를 이용해서 설정할 수 있음

• 프레임의 형태를 설정하는 상수

Constant	Value	Description
<code>QFrame::NoFrame</code>	0	<code>QFrame</code> draws nothing
<code>QFrame::Box</code>	0x0001	<code>QFrame</code> draws a box around its contents
<code>QFrame::Panel</code>	0x0002	<code>QFrame</code> draws a panel to make the contents appear raised or sunken
<code>QFrame::StyledPanel</code>	0x0006	draws a rectangular panel with a look that depends on the current GUI style. It can be raised or sunken.
<code>QFrame::HLine</code>	0x0004	<code>QFrame</code> draws a horizontal line that frames nothing (useful as separator)
<code>QFrame::VLine</code>	0x0005	<code>QFrame</code> draws a vertical line that frames nothing (useful as separator)
<code>QFrame::WinPanel</code>	0x0003	draws a rectangular panel that can be raised or sunken like those in Windows 2000. Specifying this shape sets the line width to 2 pixels. WinPanel is provided for compatibility. For GUI style independence we recommend using <code>StyledPanel</code> instead.

• 프레임의 그림자를 설정하는 상수

Constant	Value	Description
<code>QFrame::Plain</code>	0x0010	the frame and contents appear level with the surroundings; draws using the palette <code>QPalette::WindowText</code> color (without any 3D effect)
<code>QFrame::Raised</code>	0x0020	the frame and contents appear raised; draws a 3D raised line using the light and dark colors of the current color group
<code>QFrame::Sunken</code>	0x0030	the frame and contents appear sunken; draws a 3D sunken line using the light and dark colors of the current color group

• 프레임의 형태와 그림자

0				1				2				3				<code>lineWidth()</code>
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	<code>midLineWidth()</code>
																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

- QSplitter을 이용해 수평 방향으로 쪼개고, 왼쪽에는 midleft, 오른쪽에는 midright 프레임 위젯을 넣어줌
- 수직 방향으로 쪼개고, top, splitter1, bottom 3개의 프레임 위젯을 넣어줌
- splitter2 위젯을 수평 박스 레이아웃에 넣어주고, 전체의 레이아웃으로 설정함

(10) QGroupBox

- 그룹 박스는 상단 타이틀과 단축키를 제공하며, 그 안에 다양한 위젯들을 나타냄
- QGroupBox 클래스는 제목과 제목의 정렬을 설정하도록 해줌
- 그룹 박스는 체크 가능하도록 설정할 수 있는데, 그룹 박스의 체크 여부에 따라 내부 위젯들이 사용 가능하거나 불가능해짐
- 공간 절약위해 flat 속성을 활성화할 수 있으며, 프레임의 좌우, 아래쪽 가장자리가 없게 표시됨

```
#QGroupBox

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QGroupBox, QRadioButton,
QCheckBox, QPushButton, QMenu, QGridLayout, QVBoxLayout
class MyApp(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        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)

        self.setWindowTitle('Box Layout')
        self.setGeometry(300, 300, 480, 320)
        self.show()

    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

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

        radio1 = QRadioButton('Radio1')
        radio2 = QRadioButton('Radio2')
        radio3 = QRadioButton('Radio3')
        radio1.setChecked(True)
        checkbox=QCheckBox('Independent Checkbox')
        checkbox.setChecked(True)
```

```

vbox = QVBoxLayout()
vbox.addWidget(radio1)
vbox.addWidget(radio2)
vbox.addWidget(radio3)
vbox.addWidget(checkbox)
vbox.addStretch(1)
groupbox.setLayout(vbox)

return groupbox

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

    pushbutton=QPushButton('Normal Button')
    togglebutton=QPushButton('Toggle Button')
    togglebutton.setCheckable(True)
    togglebutton.setChecked(True)
    flatbutton=QPushButton('Flat Button')
    flatbutton.setFlat(True)
    popupbutton=QPushButton('Popup Button')
    menu=QMenu(self)
    menu.addAction('First Item')
    menu.addAction('Second Item')
    menu.addAction('Third Item')
    menu.addAction('Fourth Item')
    popupbutton.setMenu(menu)

    vbox=QVBoxLayout()
    vbox.addWidget(pushbutton)
    vbox.addWidget(togglebutton)
    vbox.addWidget(flatbutton)
    vbox.addWidget(popupbutton)
    vbox.addStretch(1)
    groupbox.setLayout(vbox)

return groupbox

if __name__=='__main__':
    app=QApplication(sys.argv)
    ex=MyApp()
    sys.exit(app.exec_())

```

- 그리드 레이아웃을 이용해 그룹 박스를 배치함, 각 메소드를 통해 만들어지는 그룹 박스가 addWidget()을 통해 각 위치로 배치됨
- createFirstExclusiveGroup()은 배타적인 라디오 버튼을 갖는 그룹 박스를 만들
- QGroupBox()를 이용해 먼저 그룹 박스를 하나 만들고, 버튼을 만든 다음 수직 박스 레이아웃을 통해 배치함
- 마지막으로 수직 박스 레이아웃(vbox)을 그룹 박스의 레이아웃으로 설정함, 세 개의 라디오 버튼을 만들고 수직으로 배치함
- createSecondExclusiveGroup()은 세 개의 라디오 버튼과 한 개의 체크 박스를 갖는 그룹 박스를 만들, setCheckable()을 이용해 groupbox도 선택 가능하도록 할 수 있음
- createNonExclusiveGroup()은 배타적이지 않은 체크 박스를 갖는 그룹 박스를 만들
- setFlat()을 이용해 그룹박스를 평평하게 보이도록 함
- createPushButtonGroup()은 여러 개의 푸시 버튼을 갖는 그룹 박스를 만들
- setCheckable()과 setChecked()을 이용해 그룹박스를 선택 가능하게, 그리고 실행했을 때 선택되어 있도록 설정함