3. 위젯(2)

(11) QTabWidget

- 탭은 프로그램 안의 구성요소들이 많은 면적을 차지하지 않으면서, 그것들을 카테고리에 따라 분류할 수 있기 때문에 유용하게 사용될 수 있음

```
#QTabWidget
import sys
from PyQt5.QtWidgets import  QApplication, QWidget, QTabWidget, QVBoxLayout
class MyApp(QWidget):

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

def initUI(self):
tab1=QWidget()
tab2=QWidget()

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

vbox=QVBoxLayout()
vbox.addWidget(tabs)

self.setLayout(vbox)

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

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

- 각 탭에 위치할 두 개의 위젯을 만듦
- QTabWidget()을 이용해 탭을 만들어주고, addTab()을 이용해 'Tab1'과 'Tab2'를 tabs에 추가함
- 수직 박스 레이아웃을 하나 만들어 탭 위젯(tabs)를 넣어줌
- 수직 박스를 위젯의 레이아웃으로 설정함

(12) QPixMap

・QPixMap은 이미지를 다룰 때 사용되는 위젯임

・지원하는 파일 형식

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

```
#QPixmap
import sys
from PyQt5.QtWidgets import  QApplication, QWidget, QLabel, QVBoxLayout
from PyQt5.QtGui import QPixmap
from PyQt5.QtCore import Qt

class MyApp(QWidget):

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

def initUI(self):
pixmap=QPixmap('이미지이름.확장자')
lbl_img=QLabel()
lbl_img.setPixmap(pixmap)
lbl_size=QLabel('Width: '+str(pixmap.width())+', Height: '+str(pixmap.height()))
lbl_size.setAlignment(Qt.AlignCenter)

vbox=QVBoxLayout()
vbox.addWidget(lbl_img)
vbox.addWidget(lbl_size)
self.setLayout(vbox)

self.setWindowTitle('QPixmap')
self.move(300, 300)
self.show()


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

・파일 이름을 입력해주고, QPixmap 객체(pixmap)를 하나 만듦

・라벨을 하나 만들고, setPixmap을 이용해 pixmap을 라벨에 표시될 이미지로 설정함

・width()와 height()는 이미지의 너비, 높이를 반환함

・너비, 높이를 표시하는 라벨(lbl_size)을 하나 만들고, setAlignment를 이용해 가운데로 정렬함

・수평 박스 레이아웃을 하나 만들고, 라벨을 배치함

・setLayout()을 이용해 수평 박스를 창의 레이아웃으로 지정해줌

(13) QCalendarWidget

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

```
#QCalendarWidget
import sys
from PyQt5.QtWidgets import  QApplication, QWidget, QLabel, QVBoxLayout,
QCalendarWidget
from PyQt5.QtCore import QDate

class MyApp(QWidget):

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

def initUI(self):
cal=QCalendarWidget(self)
cal.setGridVisible(True)
cal.clicked[QDate].connect(self.showDate)

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

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

self.setLayout(vbox)

self.setWindowTitle('QCalenderWidget')
self.setGeometry(300, 300, 400, 300)
self.show()

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

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

- QCalendarWidget이 객체(cal)를 하나 만듦
- setGridVisible을 True로 설정하면, 날짜 사이에 그리드가 표시됨
- 날짜를 클릭했을 때 showDate가 호출되도록 연결해줌
- selectedDate는 현재 선택된 날짜 정보를 갖고 있음(디폴트는 현재 날짜), 현재 날짜 정보를 라벨에 표시되도록 해줌
- 수직 박스 레이아웃을 이용해 달력과 라벨을 수직으로 배치해줌
- showDate에서, 날짜를 클릭할 때마다 라벨 텍스트가 선택한 날짜(date.toString())로 표시되도록 함

(14) QSpinBox

· QSpinBox 클래스는 정수를 선택, 조절하도록 하는 스핀 박스 위젯을 제공함

```
#QSpinBox
import sys
from PyQt5.QtWidgets import  QApplication, QWidget, QLabel, QSpinBox,
QVBoxLayout

class MyApp(QWidget):

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

def initUI(self):
self.lbl1=QLabel('QSpinBox')
self.spinbox=QSpinBox()
self.spinbox.setMinimum(-10)
self.spinbox.setMaximum(30)
self.spinbox.setSingleStep(2)
self.lbl2=QLabel('0')

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

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

self.setLayout(vbox)

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

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

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

· QSpinBox 객체(self.spinbox)를 하나 만듦

· setMinimum()과 setMaximum()을 이용해 선택 범위를 제한할 수 있으며, 최소값 0, 최대값 99가 디폴트임

· setRange()는 setMinimum()과 setMaximum()을 합쳐놓은 것과 같음

· setSingleStep()을 이용해 한 스텝을 2로 설정함, 스핀 박스의 경우 한 스텝으로 설정할 수 있는 최소값은 1임

· 스핀박스 위젯의 값이 변경될 때 발생하는 시그널(valueChanged)을 self.value_changed에 연결함

· 수직 박스 레이아웃을 이용해 라벨 두 개와 스핀 박스 위젯을 수직으로 배치하고, 전체 위젯의 레이아웃으로 설정함

· 스핀박스의 값이 변경될 때, self.lbl2의 텍스트를 스핀박스의 값(self.spinbox.value())으로 설정하도록 함

(15) QDoubleSpinBox

· QDoubleSpinBox 클래스는 실수를 선택, 조절하도록 하는 더블 스핀 박스 위젯을 제공함

```
#QDoubleSpinBox
import sys
from PyQt5.QtWidgets import  QApplication, QWidget, QLabel, QDoubleSpinBox,
QVBoxLayout

class MyApp(QWidget):

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

def initUI(self):
self.lbl1=QLabel('QDoubleSpinBox')
self.dspinbox=QDoubleSpinBox()
self.dspinbox.setRange(0, 100)
self.dspinbox.setSingleStep(0.5)
self.dspinbox.setPrefix('$ ')
self.dspinbox.setDecimals(1)
self.lbl2=QLabel('$ 0.0')

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

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

self.setLayout(vbox)

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

def value_changed(self):
self.lbl2.setText('$ '+str(self.dspinbox.value()))

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

· QDoubleSpinBox 객체(self.dspinbox)를 하나 만듦

· setRange()를 이용해 선택 범위를 제한함, 최소값 0.0, 최대값 99.99가 디폴트

· setSingleStep()을 이용해 한 스텝을 0.5로 설정함

· setPrefix()를 이용해, 숫자 앞에 올 문자를 설정함, setSuffix()는 숫자 뒤에 문자가 오도록 설정함

· setDecimals()를 이용해 소수점 아래 표시될 자리수를 정해줌

· 더블 스핀 박스의 값이 변경될 때 발생하는 시그널(valueChanged)을 self.value_changed에 연결함

· 수직 박스 레이아웃을 이용해 라벨 두 개와 더블 스핀 박스 위젯을 수직으로 배치하고, 전체 위젯의 레이아웃으로 설정함

· 더블 스핀 박스의 값이 변경될 때, self.lbl2의 텍스트를 더블 스핀 박스의 값(self.dspinbox.value())으로 설정함

(16) QDateEdit

- QDateEdit 위젯은 사용자에게 날짜를 선택, 편집하도록 할 때 사용함

```
#QDateEdit
import sys
from PyQt5.QtWidgets import  QApplication, QWidget, QLabel, QDateEdit,
QVBoxLayout
from PyQt5.QtCore import QDate

class MyApp(QWidget):

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

def initUI(self):
lbl=QLabel('QDateEdit')

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

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

self.setLayout(vbox)

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

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

- QDateEdit 클래스를 이용해 날짜 편집 위젯을 하나 만듦
- setDate에 QDate.currentDate()를 입력해 프로그램이 실행되는 현재 날짜를 표시함
- setMinimumDate와 setMaximumDate를 이용하면 날짜의 범위를 제한할 수 있음
- setDateRange는 setMinimumDate와 setMaximumDate를 동시에 사용하는 것과 같음
- 수직 박스 레이아웃을 이용해 라벨과 날짜 편집 위젯을 수직으로 배치하고, 전체 위젯의 레이아웃으로 설정함

(17) QTimeEdit

- QTimeEdit 위젯은 사용자에게 시간을 선택, 편집하도록 할 때 사용함

```
#QTimeEdit
import sys
from PyQt5.QtWidgets import  QApplication, QWidget, QLabel, QTimeEdit,
QVBoxLayout
from PyQt5.QtCore import QTime

class MyApp(QWidget):

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

def initUI(self):
lbl=QLabel('QTimeEdit')

timeedit=QTimeEdit(self)
timeedit.setTime(QTime.currentTime())
timeedit.setTimeRange(QTime(3, 00, 00), QTime(23, 30, 00))
timeedit.setDisplayFormat('hh:mm:ss')

vbox=QVBoxLayout()
vbox.addWidget(lbl)
vbox.addWidget(timeedit)
vbox.addStretch()

self.setLayout(vbox)

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


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

- QTimeEdit 클래스를 이용해 시간 편집 위젯(timeedit)을 하나 만듦
- setTime에 QTime.currentTime()를 입력해 프로그램이 실행되는 현재 시간을 표시함
- setTimeRange를 이용하면 사용자가 선택할 수 있는 시간으 범위를 제한할 수 있음
- setDisplayFormat을 이용해 시간의 형식을 설정함
- 수직 박스 레이아웃을 이용해 라벨과 시간 편집 위젯을 수직으로 배치하고, 전체 위젯의 레이아웃으로 설정함

(18) QDateTimeEdit

- QDateTimeEdit 위젯은 사용자에게 날짜와 시간을 선택, 편집하도록 할 때 사용함

```
#QDateTimeEdit
import sys
from PyQt5.QtWidgets import  QApplication, QWidget, QLabel, QDateTimeEdit,
QVBoxLayout
from PyQt5.QtCore import QDateTime

class MyApp(QWidget):

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

def initUI(self):
lbl=QLabel('QDateTimeEdit')

datetimeedit=QDateTimeEdit(self)
datetimeedit.setDateTime(QDateTime.currentDateTime())
datetimeedit.setDateTimeRange(QDateTime(1900, 1, 1, 00, 00, 00),
QDateTime(2100, 1, 1, 00, 30, 00))
datetimeedit.setDisplayFormat('yyyy.MM.dd hh:mm:ss')

vbox=QVBoxLayout()
vbox.addWidget(lbl)
vbox.addWidget(datetimeedit)
vbox.addStretch()

self.setLayout(vbox)

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


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

- QDateTimeEdit 클래스를 이용해 날짜, 시간 편집 위젯(datetimeedit)을 하나 만듦
- setDateTime에 QDateTime.currentDateTime()을 입력해 프로그램이 실행되는 현재 날짜와 시간으로 표시되도록 함
- setDateTimeRange를 이용하면 사용자가 선택할 수 있는 날짜와 시간의 범위를 제한함
- setDisplayFormat을 이용해 날짜와 시간의 형식을 설정함
- 수직 박스 레이아웃을 이용해 라벨과 날짜, 시간 편집 위젯을 수직으로 배치하고, 전체 위젯의 레이아웃으로 설정함

(19) QTextBrowser

- QTextBrowser 클래스는 하이퍼텍스트 네비게이션을 포함하는 리치 텍스트(서식있는 텍스트) 브라우저를 제공함
- 이 클래스는 읽기 전용이며, QTextEdit의 확장형으로 하이퍼텍스트 문서의 링크들을 사용할 수 있음
- 편집 가능한 리치 텍스트 편집기를 사용하기 위해서는 QTextEdit을 사용해야 함
- 하이퍼텍스트 네비게이션이 없는 텍스트 브라우저를 사용하기 위해서는 QTextEdit을 setReadOnly()를 사용해 편집이 불가능해도록 해줌
- 짧은 리치 텍스트를 표시하기 위해서는 QLabel을 사용할 수 있음

```
#QTextBrowser
import sys
from PyQt5.QtWidgets import  QApplication, QWidget, QLineEdit, QTextBrowser,
QPushButton, QVBoxLayout
class MyApp(QWidget):

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

def initUI(self):
self.le=QLineEdit()
self.le.returnPressed.connect(self.append_text)

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

self.clean_btn=QPushButton('Clear')
self.clean_btn.pressed.connect(self.clear_text)

vbox=QVBoxLayout()
vbox.addWidget(self.le, 0)
vbox.addWidget(self.tb, 1)
vbox.addWidget(self.clean_btn, 2)

self.setLayout(vbox)

self.setWindowTitle('QTextBrowser')
self.setGeometry(300, 300, 300, 300)
self.show()

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

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

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

- 줄 편집기를 하나 만듦
- Enter키를 누르면 append_text가 호출됨
- QTextBrowser() 클래스를 이용해 텍스트 브라우저를 하나 만듦

- setAcceptRichText()를 True로 설정해주면, 서식 있는 텍스트(Rich text)를 사용할 수 있음
- setOpenExternalLinks()를 True로 설정해주면, 외부 링크로의 연결이 가능함
- clear_btn을 클릭하면, clear_text가 호출됨
- append_text는 줄 편집기에 작성된 텍스트(self.le.text())를 텍스트 브라우저(self.tb)에 append해주는 기능을 함
- 텍스트가 텍스트 브라우저에 추가되면, clear을 이용해 줄 편집기에 있던 텍스트는 삭제됨
- clear_text가 호출되면, clear을 이용해 텍스트 브라우저(self.tb)에 있던 텍스트를 없애줌

(20) QTextEdit

- QTextEdit 클래스는 플레인 텍스트(plain text)와 리치 텍스트(rich text)를 모두 편집하고 표시할 수 있는 편집기를 제공함

```
#QTextEdit
import sys
from PyQt5.QtWidgets import  QApplication, QWidget, QLabel, QTextEdit,
QVBoxLayout
class MyApp(QWidget):

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

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

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

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

self.setLayout(vbox)

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

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

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

- QTextEdit() 클래스를 이용해 텍스트 편집기를 하나 만듦
- setAcceptRichText를 False로 하면, 모두 플레인 텍스트로 인식함
- 텍스트 편집기의 텍스트가 수정될 때마다 text_changed가 호출됨
- clear_btn을 클릭하면, clear_text가 호출됨
- 수직 박스 레이아웃을 이용해, 두 개의 라벨과 하나의 텍스트 편집기를 수직 방향으로 배치함
- text_changed가 호출되면, toPlainText()를 이용해 텍스트 편집기(self.te)에 있던 텍스트를 text 변수에 저장함
- split()은 문자열의 단어들을 리스트 형태로 바꿔줌
- setText()를 이용해 두 번째 라벨에 단어수를 표시함