# 4. 다이얼로그 (Dialog)

- 다이얼로그는 대화창이라고도 부르며 GUI 프로그래밍의 필수 요소임
- 사용자가 어플리케이션 안에서 어플리케이션과 '대화'하는데 사용됨
- 즉 다이얼로그는 사용자가 데이터를 입력, 수정하거나, 어플리케이션의 설정을 변경하는 등의 작업을 하는데 사용됨

## (1) QInputDialog

- 입력 다이얼로그는 사용자가 간단한 값을 입력할 때 사용하는 다이얼로그
- 입력값은 숫자, 문자열, 리스트에서 선택한 항목 등이 될 수 있음

```
#QInputDialog

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

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

def initUI(self):
self.btn=QPushButton('Dialog', self)
self.btn.move(30, 30)
self.btn.clicked.connect(self.showDialog)

self.le=QLineEdit(self)
self.le.move(120, 35)

self.setWindowTitle('Input dialog')
self.setGeometry(300, 300, 300, 200)
self.show()

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

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

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

- QInputDialog.getText를 통해 입력 대화창이 나타남. 두번째 매개변수는 대화창의 타이틀, 세번째 매개변수는 대화창 안에 보여질 메세지를 의미함.
- 입력 다이얼로그는 입력한 텍스트와 불리언 값을 반환함
- 텍스트를 입력한 후 'OK' 버튼을 누르면 불리언 값은 True가 됨
- 입력한 값을 setText()를 통해 줄 편집 위젯에 표시되도록 함

## (2) QColorDialog

· 색상을 선택할 수 있는 다이얼로그

```
#QColorDialog

import sys
from PyQt5.QtWidgets import  QApplication, QWidget, QPushButton, QFrame,
QColorDialog
from PyQt5.QtGui import QColor
class MyApp(QWidget):

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

def initUI(self):

col=QColor(0, 0, 0)

self.btn=QPushButton('Dialog', self)
self.btn.move(30, 30)
self.btn.clicked.connect(self.showDialog)

self.frm=QFrame(self)
self.frm.setStyleSheet('QWidget { background-color: %s }'%col.name())
self.frm.setGeometry(130, 35, 100, 100)

self.setWindowTitle('Color dialog')
self.setGeometry(300, 300, 250, 180)
self.show()

def showDialog(self):
col=QColorDialog.getColor()

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

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

· QColor를 사용해 배경색을 검정색으로 만듦

· QColorDialog를 띄우고, getColor()를 통해 색상을 저장함

· 색상을 선택하고 'OK' 버튼을 누르면, col.isValid()의 불리언 값은 True가 됨

· 선택한 색상이 프레임의 배경색으로 설정됨

## (3) QFontDialog

- 폰트를 선택할 수 있게 해주는 다이얼로그

```
#QFontDialog

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

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

def initUI(self):
btn=QPushButton('Dialog', self)
btn.setSizePolicy(QSizePolicy.Fixed, QSizePolicy.Fixed)
btn.move(20, 20)
btn.clicked.connect(self.showDialog)

vbox=QVBoxLayout()
vbox.addWidget(btn)

self.lbl=QLabel('The quick brown fox jumps over the lazy dog', self)
self.lbl.mopve(130, 20)

vbox.addWidget(self.lbl)
self.setLayout(vbox)

self.setWindowTitle('Font Dialog')
self.setGeometry(300, 300, 250, 180)
self.show()

def showDialog(self):
font, ok=QFontDialog.getFont()

if ok:
self.lbl.setFont(font)

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

- 폰트 다이얼로그를 띄우고, getFont()를 사용해 선택한 폰트와 불리언 값을 반환받음
- 'OK' 버튼을 클릭하면 True를 반환함
- setFont()를 사용해 선택한 폰트를 라벨의 폰트로 설정해줌

## (4) QFileDialog

- 사용자가 파일 또는 경로를 선택할 수 있도록 하는 다이얼로그
- 사용자는 선택한 파일을 열어서 수정하거나 저장할 수 있음

```
#QFileDialog

import sys
from PyQt5.QtWidgets import  QApplication, QMainWindow, QTextEdit, QAction,
QFileDialog
from PyQt5.QtGui import QIcon
class MyApp(QMainWindow):

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

def initUI(self):
self.textEdit=QTextEdit()
self.setCentralWidget(self.textEdit)
self.statusBar()

openFile=QAction(QIcon('이미지이름.확장자'), 'Open', self)
openFile.setShortcut('Ctrl+O')
openFile.setStatusTip('Open New File')
openFile.triggered.connect(self.showDialog)

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

self.setWindowTitle('File Dialog')
self.setGeometry(300, 300, 300, 200)
self.show()

def showDialog(self):
fname=QFileDialog.getOpenFileName(self, 'Open file', './')

if fname[0]:
f=open(fname[0], 'r')

with f:
data=f.read()
self.textEdit.setText(data)


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

- QFileDialog를 띄우고, getOpenFileName()을 이용해 파일을 선택함. 세번째 매개변수를 통해 기본 경로를 설정할 수 있고, 기본적으로 모든 파일(*)을 열도록 되어있음
- 선택한 파일을 읽어서, setText()를 통해 텍스트 편집 위젯에 불러옴

## (5) QMessageBox

- QMessageBox 클래스는 사용자에게 정보를 제공하거나 질문과 대답을 할 수 있는 대화창 제공함, 흔히 어떤 동작에 대해 확인이 필요한 경우에 메세지 박스 사용
- 사용자에게 상황을 설명하는 기본 텍스트를 표시하고, 정보를 전달하거나 사용자의 의사를 묻는 텍스트를 표시함

```
#QMessageBox

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

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

def initUI(self):
self.setWindowTitle('QMessageBox')
self.setGeometry(300, 300, 300, 200)
self.show()

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

if reply==QMessageBox.Yes:
event.accept()

else:
event.ignore()


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

- QWidget을 종료할 때, QCloseEvent가 생성되어 위젯에 전달되며, 위젯의 동작을 수정하기 위해 closeEvent() 이벤트 핸들러를 수정해야함
- QMessageBox.question의 두번째 매개변수는 타이틀바에 나타날 문자열, 세번째 매개변수는 대화창에 나타날 문자열을 입력함. 네번째에는 대화창에 보여질 버튼의 종류를 입력하고, 마지막으로 디폴트로 선택될 버튼을 설정해줌.
- 반환값은 reply 변수에 저장됨.
- 'Yes'를 클릭했을 경우, 이벤트를 받아들이고 위젯을 종료함
- 'No'를 클릭했을 경우, 이벤트를 무시하고 위젯을 종료하지 않음

# 5. 시그널과 슬롯 (Signal & Slot)

· PyQt에서는 이벤트 처리에 있어서 시그널과 슬롯이라는 독특한 매커니즘을 사용함

## (1) 연결하기

```
#연결하기

import sys
from PyQt5.QtWidgets import  QApplication, QWidget, QLCDNumber, QDial,
QVBoxLayout
class MyApp(QWidget):

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

def initUI(self):
lcd=QLCDNumber(self)
dial=QDial(self)

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

dial.valueChanged.connect(lcd.display)

self.setWindowTitle('Signal and Slot')
self.setGeometry(300, 300, 200, 200)
self.show()


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

· QLCDNumber 위젯은 LCD 화면과 같이 숫자를 표시함

· QDial은 다이얼을 회전해서 값을 조절하는 위젯

· 수직 박스 레이아웃을 하나 만들어 QLCDNumber와 QDial 위젯을 넣고, MyApp 위젯의 레이아웃으로 설정함

· QDial 위젯의 valueChanged 시그널을 lcd의 display 슬롯에 연결함. display 슬롯은 숫자를 받아서 QLCD-Number 위젯에 표시하는 역할을 함

· 시그널을 보내는 객체인 송신자(sender)는 dial, 시그널을 받는 객체인 수신자(receiver)는 lcd임

· 슬롯(slot)은 시그널에 어떻게 반응할지를 구현한 메소드임

## (2) 이벤트 핸들러 만들기

- PyQt에서 이벤트(시그널) 처리를 할 때 사용되는 함수를 이벤트 핸들러(슬롯)라고 함

```
#이벤트 핸들러 만들기

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

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

def initUI(self):
lcd=QLCDNumber(self)
dial=QDial(self)
btn1=QPushButton('Big', self)
btn2=QPushButton('Small', self)

hbox=QHBoxLayout()
hbox.addWidget(btn1)
hbox.addWidget(btn2)

vbox=QVBoxLayout()
vbox.addWidget(lcd)
vbox.addWidget(dial)
vbox.addLayout(hbox)
self.setLayout(vbox)

dial.valueChanged.connect(lcd.display)
btn1.clicked.connect(self.resizeBig)
btn2.clicked.connect(self.resizeSmall)

self.setWindowTitle('Signal and Slot')
self.setGeometry(200, 200, 200, 250)
self.show()

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

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


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

- btn1과 btn2는 각각 resizeBig, resizeSmall 슬롯에 연결됨
- resizeBig()은 화면 크기를 가로 400px, 세로 500px로 확대, resizeSmall()은 화면 크기를 가로 200px, 세로 250px로 축소하는 기능을 가지게 됨

## (3) 이벤트 핸들러 재구성하기

· 자주 쓰이는 이벤트 핸들러

| 이벤트 핸들러 | 설명 |
|---|---|
| keyPressEvent | 키보드를 눌렀을 때 동작 |
| keyReleaseEvent | 키보드를 눌렀다 뗄 때 동작 |
| mouseDoubleClickEvent | 마우스를 더블클릭할 때 동작 |
| mouseMoveEvent | 마우스를 움직일 때 동작 |
| mousePressEvent | 마우스를 누를 때 동작 |
| mouseReleaseEvent | 마우스를 눌렀다 뗄 때 동작 |
| moveEvent | 위젯이 이동할 때 동작 |
| resizeEvent | 위젯의 크기를 변경할 때 동작 |

```
#이벤트 핸들러 재구성하기

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

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

def initUI(self):
self.setWindowTitle('Reimplementing event handler')
self.setGeometry(300, 300, 300, 200)
self.show()

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()

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

· keyPressEvent 이벤트 핸들러는 키보드의 이벤트를 입력으로 받음

· e.key()는 어떤 키를 누르거나 뗐는지를 반환함

· 'esc'키를 눌렀다면, self.close()를 통해 위젯이 종료됨

· 'F'키 또는 'N'키를 눌렀다면, 위젯의 크기가 최대화되거나 보통 크기가 됨

## (4) 이벤트 핸들러 재구성하기2

```
#이벤트 핸들러 재구성하기2

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

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

def initUI(self):
x=0
y=0

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

self.setMouseTracking(True)

self.setWindowTitle('Reimplementing event handler')
self.setGeometry(300, 300, 300, 200)
self.show()

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()

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

- x, y 값을 self.text로 저장하고, self.label의 텍스트로 설정함. 위치를 x=20, y=20만큼 이동해줌
- setMouseTracking을 True로 설정해주면, 마우스의 위치를 트래킹함. 디폴트는 False 상태이며, 마우스 버튼을 클릭하거나 뗄 때만 mouseEvent가 발생함
- 이벤트 e는 이벤트에 대한 정보를 갖고 있는 하나의 객체임
- e.x(), e.y()는 위젯 안에서 이벤트가 발생했을 때 마우스 커서으 위치를 반환함
- 만약 e.globalX(), e.globalY()로 설정해주면, 화면 전체에서 마우스 커서의 위치를 반환함
- self.label.adjustSize()로 라벨의 크기를 자동으로 조절하도록 함

## (5) 사용자 정의 시그널

```
#사용자 정의 시그널

import sys
from PyQt5.QtWidgets import  QApplication, QMainWindow
from PyQt5.QtCore import pyqtSignal, QObject

class Communicate(QObject):
closeApp=pyqtSignal()
class MyApp(QMainWindow):

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

def initUI(self):
self.c=Communicate()
self.c.closeApp.connect(self.close)

self.setWindowTitle('Emitting Signal')
self.setGeometry(300, 300, 300, 200)
self.show()

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

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

- pyqtSignal()을 가지고 Communicate 클래스의 속성으로서 closeApp이라는 시그널을 만듦
- closeApp 시그널은 MyApp 클래스의 close() 슬롯에 연결됨
- mousePressEvent 이벤트 핸들러를 사용해, 마우스를 클릭했을 때 closeApp 시그널이 방출되도록 함