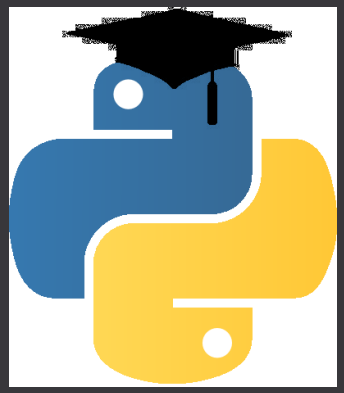


Python

PyQT pour l'IHM

Structure d'un programme QT

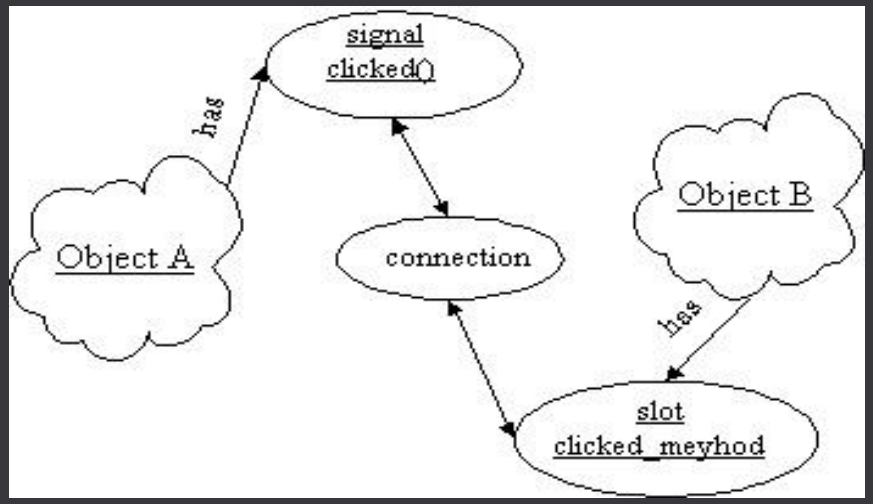
- Un programme PyQt typique contient les étapes suivante
  1. Création de l'instance de `QApplication`
  2. Création et affichage de la fenêtre principale et ses composants
  3. Connexion des signaux aux slots : gestion des évènements (QtCore)
  4. Appel de la méthode `exec__()` de l'application

Exemple :

```
# 1 - Création d'une instance QApplication
app = QApplication(args)
# 2 - Création de fenêtre
fen = QMainWindow()
fen.setWindowTitle("Demo QMainWindow")
fen.resize(40,80)
fen.show()
# 3 - Connexion des signaux aux slots
# TODO
# 4 - Création de fenêtre
app.exec_()
```

Signaux et slots en Python

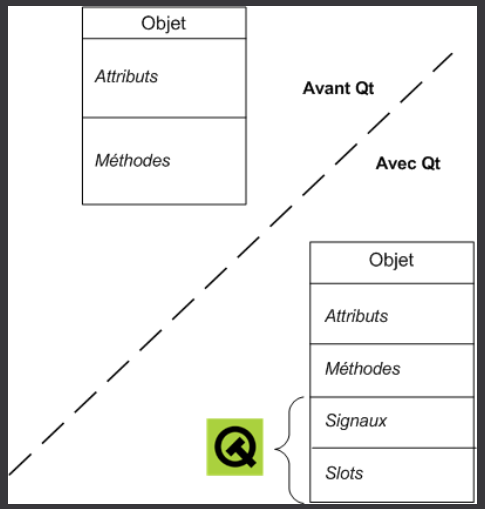
- Mécanisme utilisé dans QT pour assurer la communication entre les objets.
  - QTCore
- Un signal : c'est un message envoyé par un widget lorsqu'un évènement se produit. Exemple : on a cliqué sur un bouton.
- Un slot : c'est la fonction qui est appelée lorsqu'un évènement s'est produit. On dit que le signal appelle le slot. Concrètement, un slot est une méthode d'une classe.




```
PyQt4.QtCore.QObject.connect(emeteur, SIGNAL(), recepteur, SLOT())
```

Signaux et slots en Python

- Mécanisme utilisé dans QT pour assurer la communication entre les objets.
- Un signal : c'est un message envoyé par un widget lorsqu'un évènement se produit. Exemple : on a cliqué sur un bouton.
- Un slot : c'est la fonction qui est appelée lorsqu'un évènement s'est produit. On dit que le signal appelle le slot. Concrètement, un slot est une méthode d'une classe.



Signaux et slots en Python

Principe :

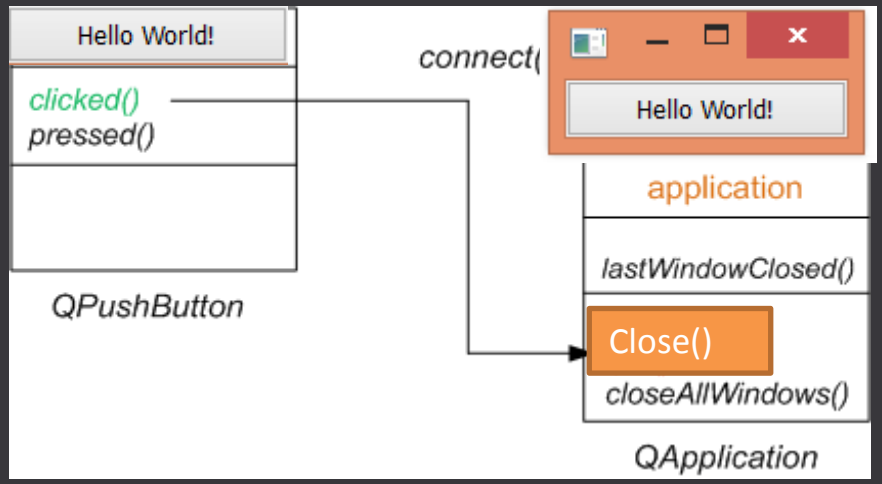
- La connexion : la méthode connect() de l'objet Qobject

Syntaxe :

```
QObject.connect(emeteur, SIGNAL(), recepteur, SLOT())
```

Elle prend 4 arguments :

1. l'objet qui émet le signal.
2. Le nom du signal que l'on veut "intercepter".
3. L'objet qui contient le slot récepteur.
4. Le nom du slot qui doit s'exécuter lorsque le signal se produit



Signaux et slots en Python

- Exemples :

1. Fermer la fenêtre en cliquant sur un bouton
2. Afficher ‘Hello’ en cliquant sur un bouton

```
from PyQt4.QtGui import *
from PyQt4.QtCore import *
import sys
class MyForm(QMainWindow):
   def __init__(self, parent=None):
       super().__init__(parent)
       the_button = QPushButton('Hello')
       self.connect(the_button, SIGNAL('clicked()'), self.do)
       self.setCentralWidget(the_button)
   def do(self):
       print('Hello')
```

Signaux et slots en Python

Principe :

- La connexion : la méthode connect() de l'objet QObject

Nouvelle Syntaxe (à partir de PyQt4.5)

```
sender.signalName.connect(receiver.slotName)
```

```
...

the_button = QPushButton('Close')
the_button.clicked.connect(self.close)
# Pour appeler une méthode hello()
the_button.clicked.connect(self.hello)
```

Signaux et slots en Python

Remarques

- Un signal peut se connecter à plusieurs slots.

  → Les slots connectés sont activés dans un ordre arbitraire.
- Un signal/slot peut avoir des arguments.
  - Pour connecter un signal à un slot, le slot (méthode) doit avoir au moins les même arguments (nombre)
  - Si le signal connecté à un slot a plus de paramètres que ce slot, les paramètres supplémentaires seront ignorés
- Les arguments entre signal et slot connectés doivent avoir les même types.
- Un signal peut se connecter à un autre signal.

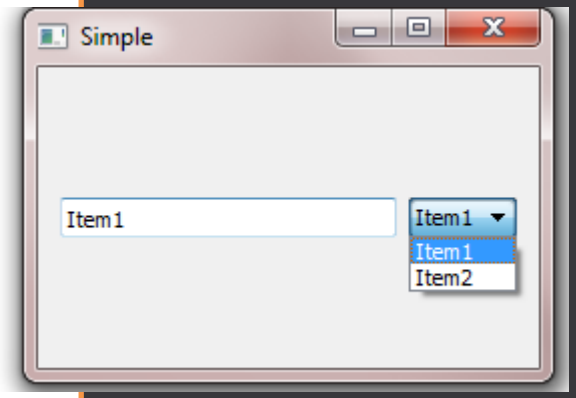
Exemple :

On ne peut connecter directement le signal valueChanged(int) du widget Qdial au slot setText(QString) du widget QLineEdit.

Signaux et slots en Python

Exemple :

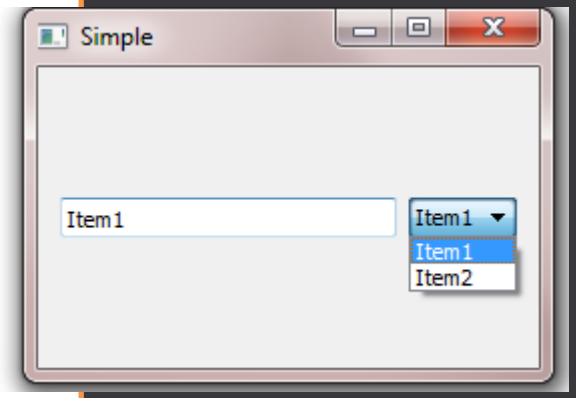
```
from PyQt4 import QtGui, QtCore
import sys
class Composant(QtGui.QWidget):
    def __init__(self,parent=None):
        QtGui.QWidget.__init__(self, parent)
        self.setWindowTitle('Simple')
        self.setGeometry(300,300,250,150)
        self.Line=QtGui.QLineEdit(' ',self)
        self.ComboBox=QtGui.QComboBox(self)
        self.ComboBox.addItem('Item1')
        self.ComboBox.addItem('Item2')
        self.grid=QtGui.QGridLayout()
        self.grid.addWidget(self.Line,0,0)
        self.grid.addWidget(self.ComboBox,0,1)
        self.setLayout(self.grid)
        self.connect(self.ComboBox,QtCore.SIGNAL('currentIndexChan
        ged(QString)'),self.Line,QtCore.SLOT('setText(QString)'))
```



Signaux et slots en Python

Exemple :

```
from PyQt4 import QtGui, QtCore
import sys
class Composant(QtGui.QWidget):
    def __init__(self,parent=None):
        Super().__init__(parent)
        self.setWindowTitle('Simple‘)
        self.setGeometry(300,300,250,150)
        self.Line=QtGui.QLineEdit(' ',self)
        self.ComboBox=QtGui.QComboBox(self)
        self.ComboBox.addItem('Item1')
        self.ComboBox.addItem('Item2')
        self.grid=QtGui.QGridLayout()
        self.grid.addWidget(self.Line,0,0)
        self.grid.addWidget(self.ComboBox,0,1)
        self.setLayout(self.grid)
        self.connect(self.ComboBox,QtCore.SIGNAL('currentIndexChan
        ged(QString)'),self.Line,QtCore.SLOT('setText(QString)'))
```



Définition signal/slot

Signal

- Utilisation de la méthode pyqtsignal() avec la liste des types de paramètres
- Emission manuelle d'un signal est réalisée par la méthode : emit()

Exemple :

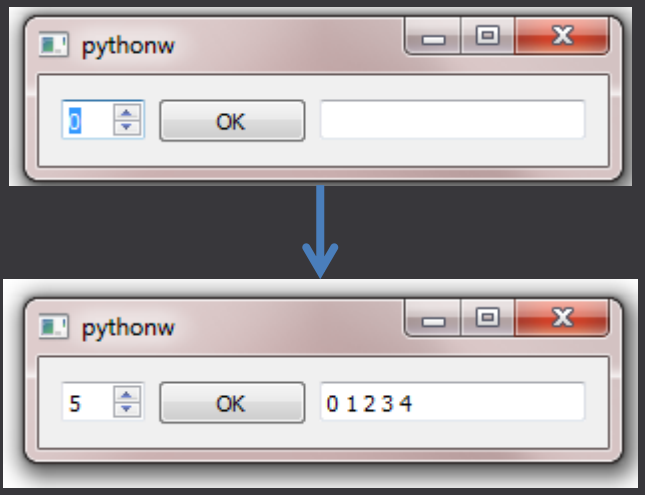
```
from PyQt4 import QtGui, QtCore
mysignal = QtCore.pyqtSignal(list, int)
mysignal.emit([1,2,3,6],3)
# error  :  mysignal.emit([1,2,3,6], '3')
```

- Le signal crée se connecte à un slot avec la même liste de types de paramètres

```
connect(emetteur, SIGNAL(``mysignal(list,int)"), methode)
```

Exemple

```
import sys
from PyQt4 import QtGui, QtCore
class MyWidget(QtGui.QWidget):
  mysignal = QtCore.pyqtSignal(list)
  def __init__(self, parent=None):
     QtGui.QWidget.__init__(self,parent)
     self.button = QtGui.QPushButton("OK", self)
     self.text=QtGui.QLineEdit()
     self.spin=QtGui.QSpinBox()
     self.grid=QtGui.QGridLayout()
     self.grid.addWidget(self.button,0,1)
     self.grid.addWidget(self.spin,0,0)
     self.grid.addWidget(self.text,0,2)
     self.setLayout(self.grid)
     self.button.clicked.connect(self.onClicked
     self.mysignal.connect(self.onPrint)
```



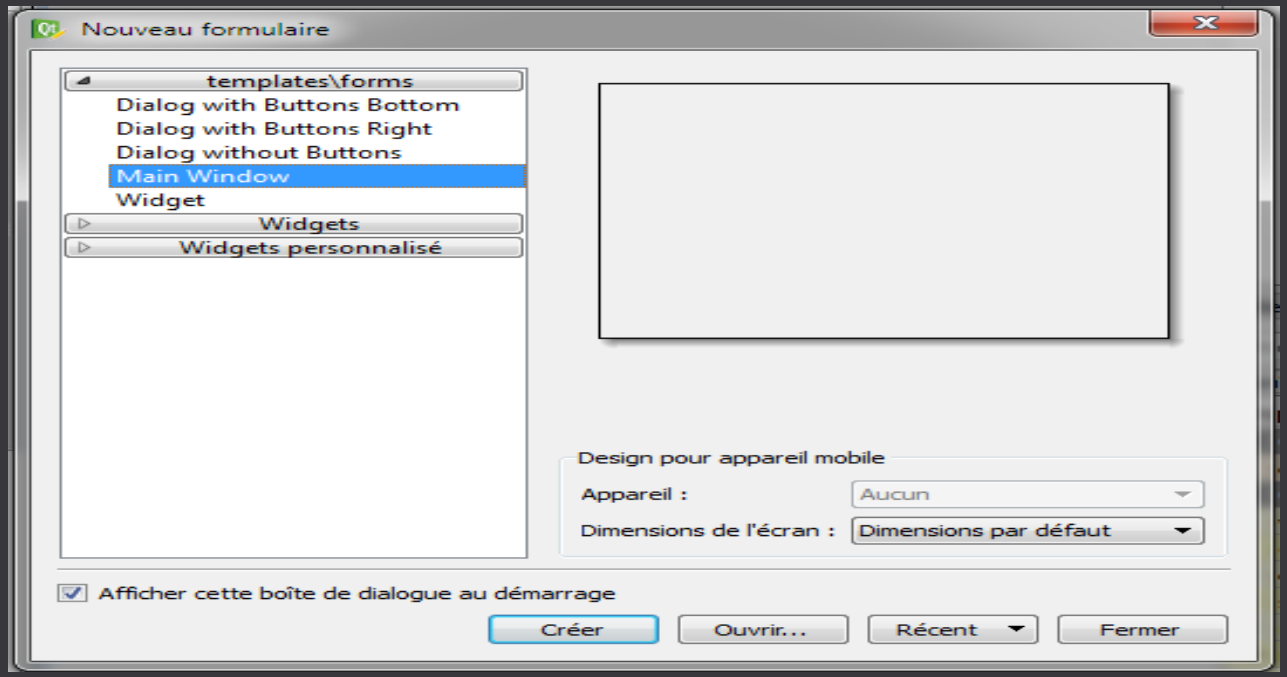

```
def onClicked(self):
   val=self.spin.value()

   self.mysignal.emit(list(range(val)))
   #self.emit(QtCore.SIGNAL('mysignal')
   #,list(range(val)))
 def onPrint(self, val):
   s= ' '
   for el in val:
      s+=str(el)+' '
   self.text.setText(s)
```

Sommaire

QTDesigner

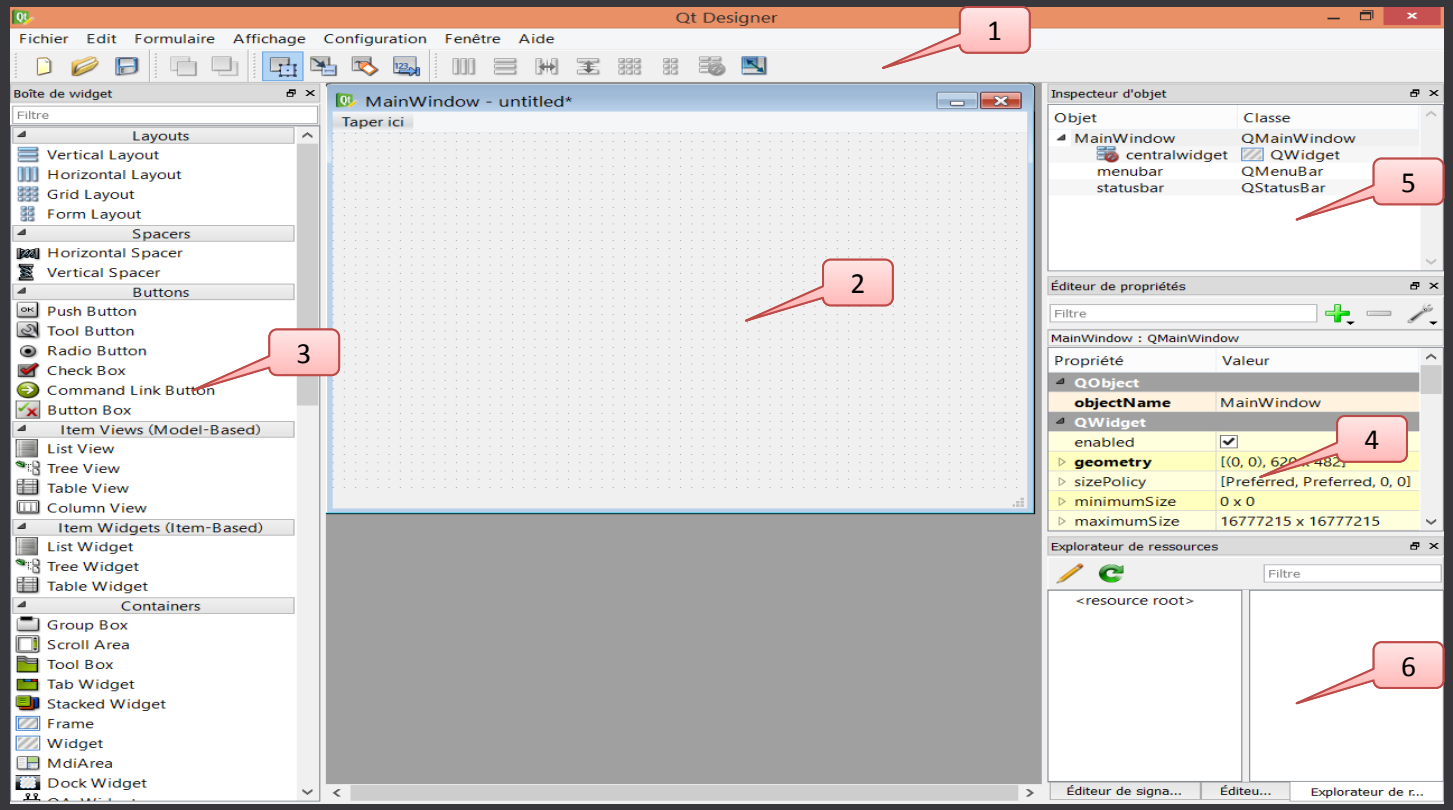
QtDesigner

- Qt Designer est un outil de design pour QT,
- Permet de lier le langage Python avec la bibliothèque Qt.
- QtDesigner est fourni avec WinPython installé sur les machines de l'école.
- Permet de réaliser une Interface Homme Machine IHM (ou GUI pour Graphic User Interface) par le biais d'un système de glisser-déposer.



QtDesigner :

- Fenêtre principale

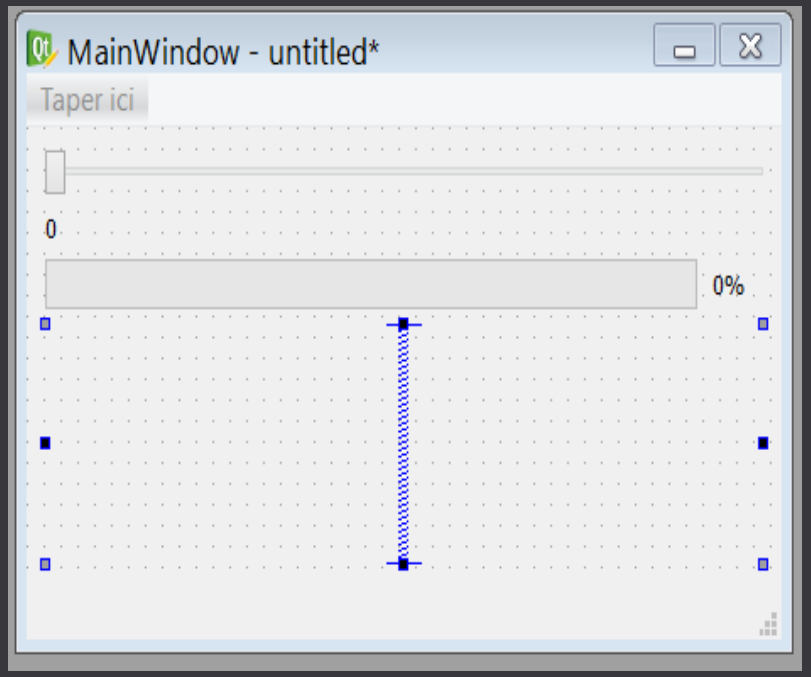


QtDesigner

- Il permet de réaliser une Interface Homme Machine IHM (ou GUI pour Graphic User Interface) par le biais d'un système de glisser-déposer

Exemple :

- Qslider
- QLabel
- QProgressBar
- QGridLayout
- Spacer (H/V)
- Barre d'outil/barre d'état

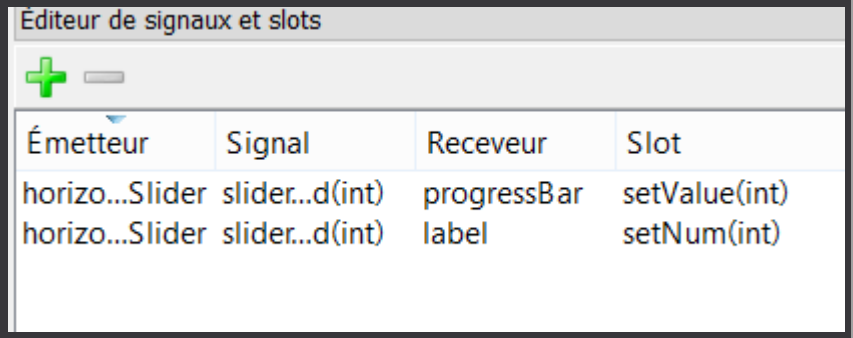


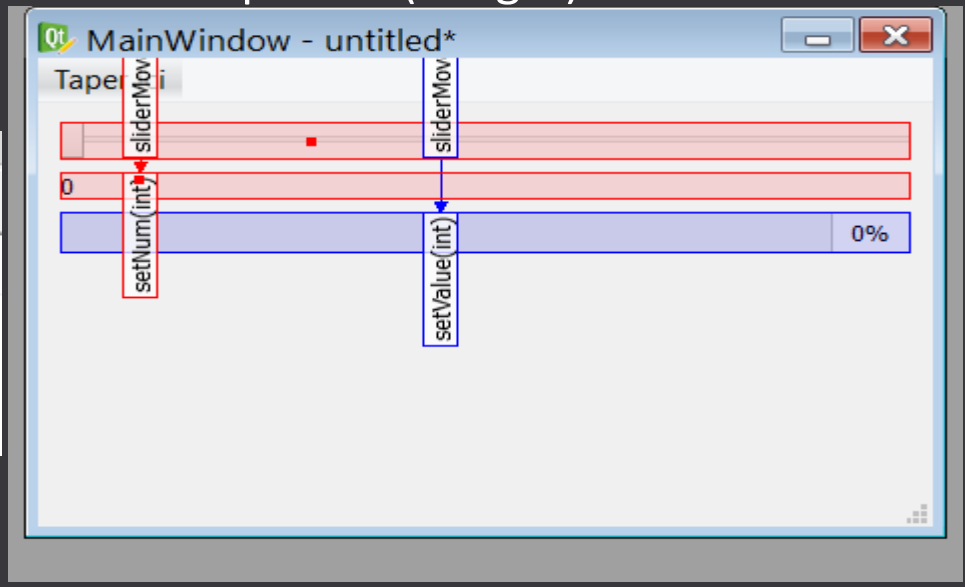
- Visualiser l'aperçu de votre fenêtre
  - CTRL+R ou F3

QtDesigner

Connexion signal/slot

- Se réalise en connectant à la sourir deux composant (widget)






```
PyQt4.QtCore.QObject.connect(emeteur, SIGNAL(), recepteur, SLOT())
```

```
sender.signalName.connect(receiver.slotName)
```

QtDesigner

- QtDesigner fournit un fichier .ui (fichier XML)
- Utilisation :

1. Par appel dynamique du fichier « .ui ». Exemple *monAppli.ui*
2. Par dérivation de la classe QmainWindow
3. Par dérivation multiples des classes QmainWindow et Ui_MainWindow

```
<property name="sizePolicy" >
<sizepolicy>
<hsizetype>1</hsizetype>
<vsizetype>1</vsizetype>
<horstretch>0</horstretch>
<verstretch>0</verstretch>
</sizepolicy>
</property>
<property name="windowTitle" >
<string>MainWindow</string>
</property>
<widget class="QWidget" name="centralwidget" >
<widget class="QPushButton" name="pushButton" >
<property name="geometry" >
<rect>
<x>80</x>
<y>130</y>
<width>75</width>
<height>23</height>
</rect>
</property>
<property name="text" >
<string>PushButton</string>
</property>
</widget>
```

QtDesigner

1. Par appel dynamique du fichier « .ui ». Exemple *monAppli.ui*
2. Par dérivation de la classe QmainWindow
3. Par dérivation multiples des classes QmainWindow et Ui_MainWindow

Transformation : via la commande *pyuic4*

>>pyuic4 MonAppli.ui -x MonAppli.py Syntaxe générale : >>pyuic4 [options] LeFichier.ui

Options	Utilité
-h ou --help	Un message d' aide apparait à la sortie
-version	Le numéro de version est écrit sur la sortie.
i N ou --indent=N	Le code source Python généré sera indenté de n espaces par défaut. Si N=0 des tabulations sont utilisées au lieu des espaces La valeur par défaut est N=4
-o Fichier ou --output=Fichier	Le source est généré dans un fichier dont il faut donner l' extension ".py"
-p ou --preview	L'interface est générée dynamiquement, et aucun source Python n'est créé.
-x ou --execute	Le code source généré comportera un petit bout de code supplémentaire pour être exécutable.

QtDesigner

Transformation : via la commande *pyuic4*

>>pyuic4 MonAppli.ui -x ui_principal.py

```
import sys from PyQt4
import QtCore, QtGui
class Ui_MainWindow(object):
     def setupUi(self, MainWindow):
         MainWindow.setObjectName("MainWindow")      )
         self.centralwidget = QtGui.QWidget(MainWindow)
         QtGui.QPushButton(self.centralwidget)
         self.pushButton.setGeometry(QtCore.QRect(80,130,75,23))
         …
if __name__ == "__main__":
    app = QtGui.QApplication(sys.argv)
    MainWindow = QtGui.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

QtDesigner

1. Appel dynamique du fichier .ui

```
…
if __name__ == "__main__":
    app = QtGui.QApplication(sys.argv)
    MainWindow = QtGui.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

```
…
if __name__ == "__main__":
    app = QtGui.QApplication(sys.argv)
    ui = uic.loadUi(« monAppli.ui")
    ui.show()
    sys.exit(app.exec_())
```

Remarque :

- Solution assez limitée
- Aucune modification n'est possible (ajout des slot, ….)

QtDesigner

1. Appel dynamique du fichier .ui
2. Par dérivation simple

```
from ui_principal import Ui_MainWindow
from PyQt4 import QtGui, uic
class MonAppli(QtGui.QMainWindow):
    def __init__(self):
        super.  init   () # Configure l'interface utilisateur.
        Self.ui = uic.loadUi(« monAppli.ui")
        # Connecte le bouton.
        self.ui.pushButton.clicked.connect(self.message)
    def message(self):
        print (’’traitement à faire le bouton" )
if __name__ == "__main__":
     app = QtGui.QApplication(sys.argv)
     window = MonAppli()
     window,show()
     sys.exit(app.exec_())
```

Remarque :

- Permet d’ajouter des objets et des méthodes
- Accéder aux Widget via une variable d’instance self.ui

QtDesigner

1. Appel dynamique du fichier .ui
2. Par dérivation simple
3. Dérivation multiple (QMainWindow, Ui_MainWindow)

```
import sys from PyQt4
import QtGui, QtCore
from ui_principal import Ui_MainWindow
class MonAppli(QtGui.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtGui.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)
        self.pushButton.clicked.connect(self.message)
    def message(self):
        print (« traitement à faire le bouton" )
if __name__ == "__main__":
    app = QtGui.QApplication(sys.argv)
    window = MonAppli()
    window,show()
    sys.exit(app.exec_())
```

Remarque :

- Permet de personnaliser la fenêtre
- Accéder directement aux Widget

A vous de jouer

Projet informatique