Python

PyQt pour les Interfaces Homme-Machine

A. Toumi
toumiab@ensta-Bretagne.fr

Sommaire

1. Création d'interface graphique (IHM)
   - Généralités
   - Composants graphiques de base
2. Présentation de QT
   - Introduction
   - Les principaux modules
   - Documentation
3. IHM avec PyQt
   - Les modules de PyQt
   - Les widgets
   - Définition de classe graphique
   - Les fenêtres
   - Signaux et slots en Python

Généralités

Intérêt :

- Présenter des informations de manière synthétique
- Permettre d'interagir avec le programme

Caractéristiques :

- Code sans difficulté technique
- Code souvent long et peu lisible
- Code pas intéressant à écrire

Solution :

- Utilisation des « boîtes à outils (toolkit graphique)»
- Programmation événementielle

Composants graphiques de base

Quelques boîtes à outils graphiques

Toolkit	Nom Python	Plate-formes	Licence
Qt	PyQt	Toutes (installée)	LGPL(Qt4)+GPL(PyQt4)/cciale
Qt	PySide	Toutes	LGPL
wxWidgets	wxPython	Toutes (installée)	LGPL
Tcl/Tk	Tkinter	Toutes (installée)	Licence Python
Gtk	pyGtk	Toutes	LGPL

Sommaire

1. Création d'interface graphique (IHM)
   - Généralités
   - Composants graphiques de base
2. Présentation de QT
   - Introduction
   - Les principaux modules
   - Documentation
3. IHM avec PyQt
   - Les modules de PyQt
   - Les widgets
   - Définition de classe graphique
   - Les fenêtres
   - Signaux et slots en Python

Introduction

- Qt est une bibliothèque C++ pour le développement d'application indépendantes des plateformes
- Qt permet la portabilité des applications qui n'utilisent que ses composants par simple recompilation du code source.
- Qt offre des composants d'interface graphique (widgets), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution, d'analyse XML, etc
- Qt supporte des bindings avec plus d'une dizaine de langages autres que le C++, comme Java, Python, Ruby, Ada, C#, Pascal, Perl, Common Lisp, etc.
- QT est disponible en licence commerciale ou libre

Les principaux modules

- *QtCore* : pour les fonctionnalités non graphiques utilisées par les autres modules : mécanisme de support pour les signaux et les slots, …
- *QtGui* : pour les composants graphiques 2D, image, text, et fonts ;
- *QtWidgets* : contient des classes qui fournissent l'ensemble de composants graphiques dans l'IHM
- *QtNetwork* : pour la programmation réseau ;
- *QtOpenGL* : 3D basée sur OpenGL ;
- *QtSql* : pour l'utilisation de base de données SQL ;
- *QtXml* : pour la manipulation et la génération de fichiers XML ;
- *QtDesigner* : pour étendre les fonctionnalités de Qt Designer, l'assistant de création d'interfaces graphiques ;
- *QtAssistant* : pour l'utilisation de l'aide de Qt ;
- QtWebKit : widget navigateur web (webkit);
- QtTest : tests unitaires
-

Consulter https://fr.wikipedia.org/wiki/Qt

La documentation

- La documentation est accessible

En ligne : *http://doc.qt.io/qt-5/*
http://pyqt.sourceforge.net/Docs/PyQt5/

Documentation de référence *: http://doc.qt.io/qt-5/qtmodules.html*
http://pyqt.sourceforge.net/Docs/PyQt5/modules.html#ref-module-index

Via le navigateur d'aide QT Assistant : la commande
>> assistant

- Documentation de la liste des classes : *http://pyqt.sourceforge.net/Docs/PyQt5/sip-classes.html*

Codes sources des exemples de démos

- Des centaines d'exemples et leurs codes sont accessibles et fournis
- *>> QT demos*

Sommaire

1. Création d'interface graphique (IHM)
   - Généralités
   - Composants graphiques de base
2. Présentation de QT
   - Introduction
   - Les principaux modules
   - Documentation
3. IHM avec PyQt
   - Les modules de PyQt
   - Les widgets
   - Définition de classe graphique
   - Les fenêtres
   - Signaux et slots en Python

Modules de PyQt5

- *QtCore* : pour les fonctionnalités non graphiques utilisées par les autres modules : mécanisme de support pour les signaux et les slots, …
- *QtWidgets* : contient des classes (composants) graphiques : QMainWindow, QApplication, QWidget, QPushButton
- *QtGui* : pour les composants graphiques ;
- *QtNetwork* , Qtbluetooth
- *QtOpenGL*
- *QtSql*
- *QtXml* /QtXmlPatterns
- *QtAssistant*
- QtWebKit
- QtTest
- …

Composants graphiques de base

Deux grandes catégories de composants :

1. Les conteneurs : peuvent contenir d'autres composants

   Exemple : zone, zone scrollable, groupement, ...

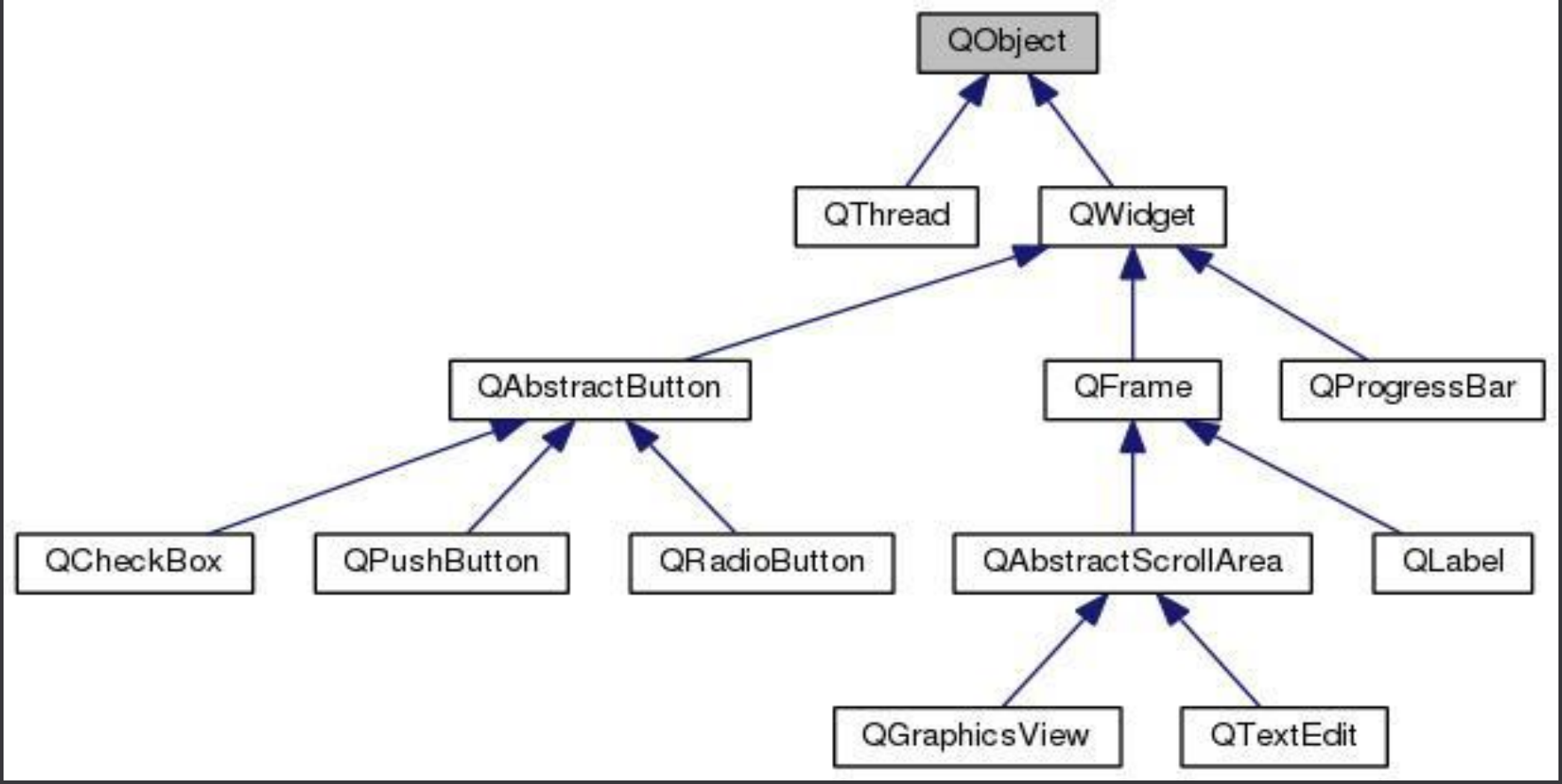
   - Les conteneurs de haut niveau (top level containers) ne peuvent pas être contenus dans d'autres conteneurs.
     Exemple : Fenêtre , boîte de dialogue
2. Autres : ne peuvent contenir d'autre composants

   Exemple : Barre de défilement, Bouton de bascule, ...

Module : QtWidgets

Les widgets

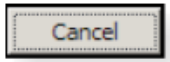
- Qwidget : la classe de base de tous les objets visibles dans l'IHM
-
-
-



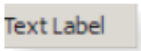
Module : QtWidgets

Qwidget

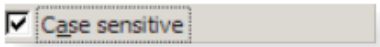
QPushButton



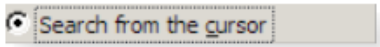
QLabel



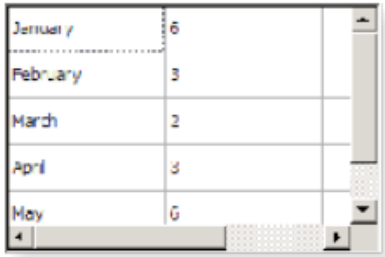
QCheckBox



QRadioButton



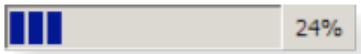
QTableView



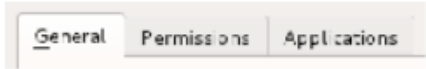
QComboBox

QSlider

QProgressBar

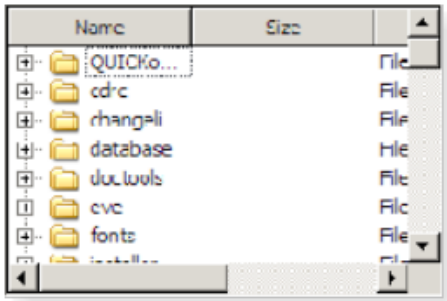


QTabBar



QMenu

QTreeView



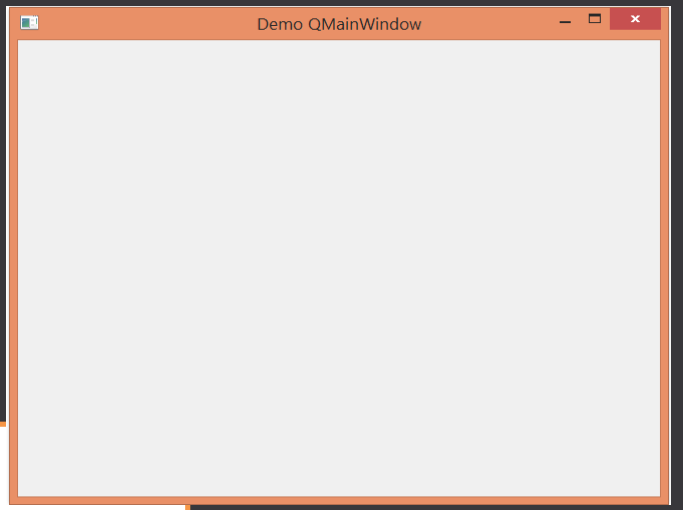
Structure d'un programme Qt

- 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
     - Connexion des signaux aux slots : gestion des évènements (QtCore)
  4. Appel de la méthode exec_() de l'application

```
from PyQt5 import import QtWidgets
# 1 - Création d'une instance QApplication
app = QtWidgets.Application(args)
# 2 - Création de fenêtre
fen = QtWidgets.QMainWindow()
fen.setWindowTitle("Demo QMainWindow")
fen.resize(40,80)
fen.show()
# 2 –a  – Connexion des signaux aux slots
 # TODO
# 4 - Création de fenêtre
app.exec_()
```

Exemple 1 : widget simple

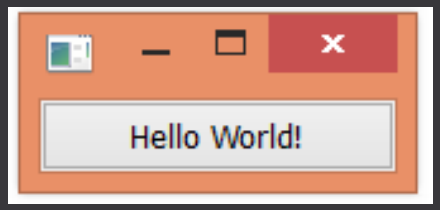
```
from PyQt5 import QtWidgets
import sys
app = QtWidgets.QApplication(sys.argv)
fen = QtWidgets.QWidget()
fen.setWindowTitle("Demo QMainWindow")
fen.resize(840,480)
fen.show()
app.exec_()
```




```
from PyQt5 import  QtWidgets
import sys
def main(args) :
  # on doit disposer d'une instance de QApplication gérant l'ensemble des widgets
  app = QtWidgets.QApplication(args)
  # une fenetre
  fen = QtWidgets.QWidget()
  # titre de la fenetre
  fen.setWindowTitle("Demo QMainWindow")
  # taille de la fenetre
  fen.resize(840,480)
  fen.show()
  app.exec_()
if __name__ == "__main__" :
  main(sys.argv)
```

Exemple 2 : création d'un bouton

Version simple :



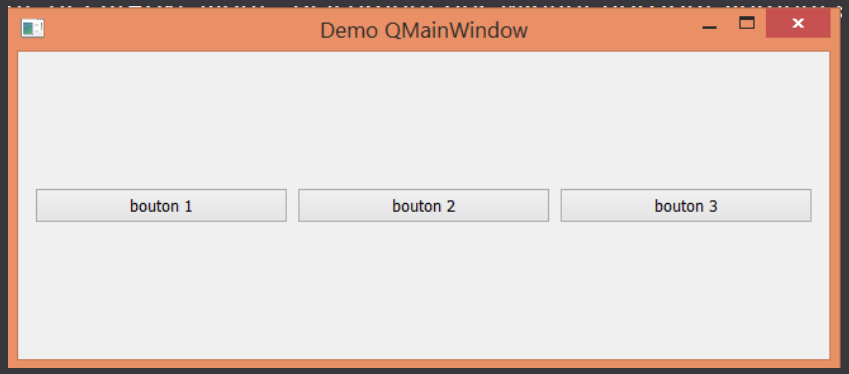

```
from PyQt5 import QtWidgets
import sys
app = QtWidgets.QApplication(sys.argv)
hello = QtWidgets.QPushButton("Hello World!", None)
hello.show()
app.exec_()
```

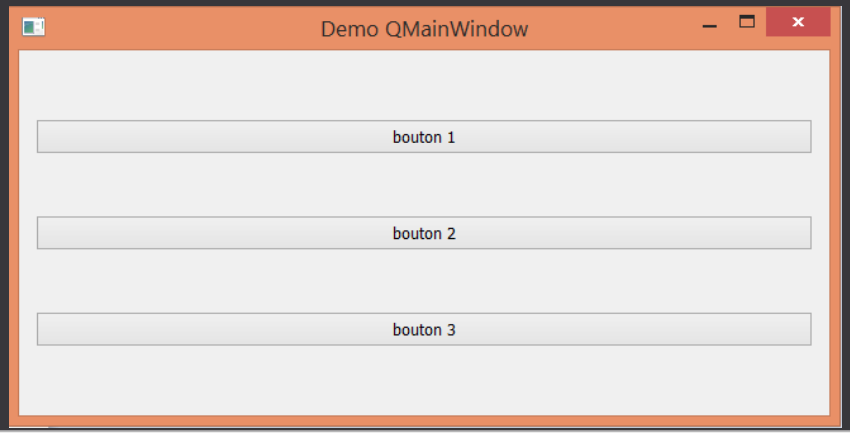
Version recommandée :

```
from PyQt5 import QtWidgets
import sys


def main(args) :
  # on doit disposer d'une instance de QApplication gérant l'ensemble des widgets
  app = QtWidgets.QApplication(args)
  # un nouveau bouton
  button = QtWidgets.QPushButton("Hello World !", None)
  # qu'on affiche
  button.show()
  app.exec_()


if __name__ == "__main__" :
  main(sys.argv)
```

Agencement des widgets (*layout*)

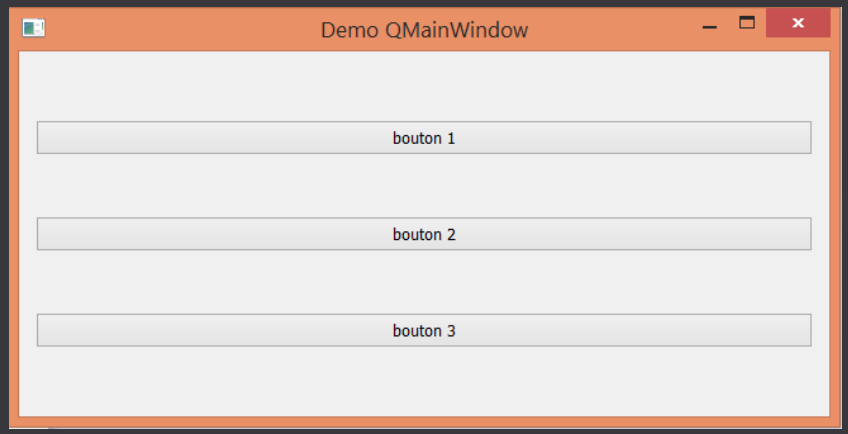
- Plusieurs types d'agencement :
  - QHBoxLayout et QVBoxLayout : placement des widgets horizontalement et verticalement
    - addWidget(widget) : ajouter un widget :
    - addSpacing(size) : ajouter d'un espace de taille fixe
    - addStretch(stretch_factor) : ajout un espace de taille extensible
  - QGridLayout : placement des widgets en grille
    - addWidget(widget, row, col) : ajouter un widget à la position (row, col)
    - addMultiCellWidget(widget, fromRow, toRow, fromCol, toCol)
  - QFormLayout : …



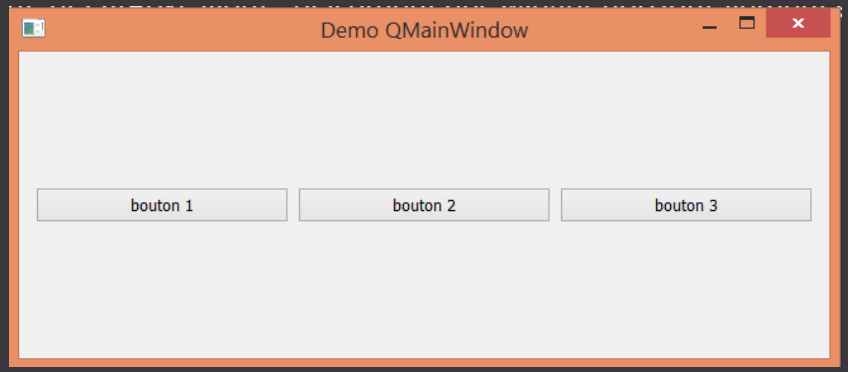


Agencement des widgets (*layout*)

```
from PyQt5 import QtWidgets
import sys

app = QtWidgets.QApplication(sys.argv)

fen = QtWidgets.QWidget()
fen.setWindowTitle("Demo QMainWindow")
fen.resize(640,480)

layout=QtWidgets.QVBoxLayout()
bouton1= QtWidgets.QPushButton("bouton 1", fen)
bouton2= QtWidgets.QPushButton("bouton 2",fen)
bouton3= QtWidgets.QPushButton("bouton 3",fen)
layout.addWidget(bouton1)
layout.addWidget(bouton2)
layout.addWidget(bouton3)

fen.setLayout(layout)
fen.show()
app.exec_()
```



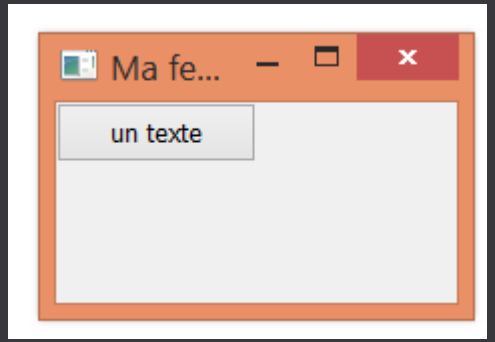
layout=QtWidgets.QHBoxLayout()

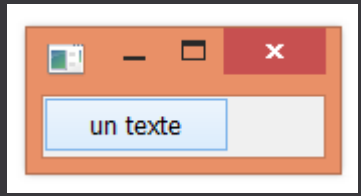


Définition d'une classe

- Classe héritant d'un QWidget , QMainWindow, QDialog
- Variables d'instance : divers composants de l'interface
- Exemple :

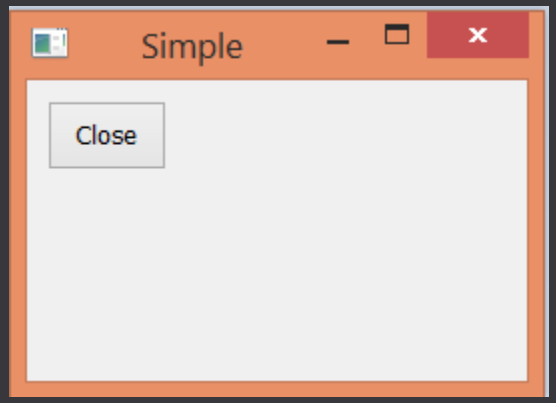
```
from PyQt5 import QtWidgets
import sys
class Fenetre(QtWidgets.QDialog):
  def __init__(self):
     super().__init__()
     self.setWindowTitle("Ma fenetre")
     self.bouton = QtWidgets.QPushButton('un texte', self)

if __name__=='__main__' :
   app = QtWidgets.QApplication(sys.argv)
   fen = Fenetre()
   fen.show()
   app.exec_()
```



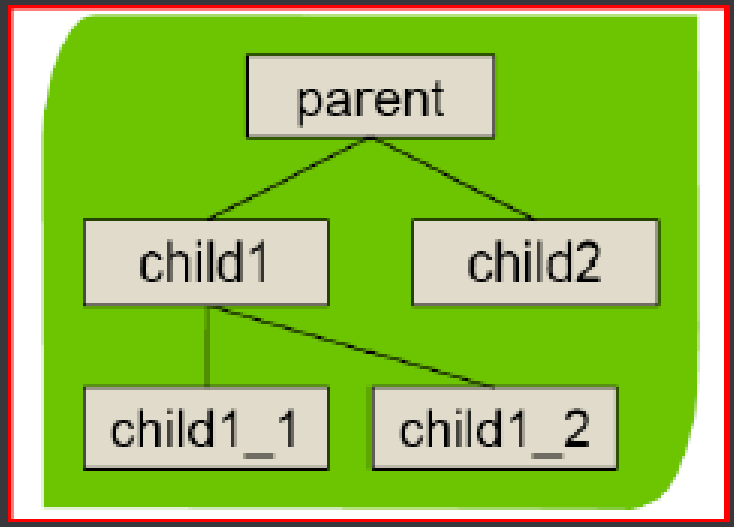


Remarques

- Toutes les instances des classes PyQt héritant de la classe QObject peuvent avoir un « parent »
- Si un objet parent est supprimé,
- tous les objets « fils » sont supprimés




```
import sys
from PyQt5 import QtWidgets
class QuitButton(QtWidgets.QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Simple')
        quit = QtWidgets.QPushButton('Close', self)
        quit.setGeometry(10, 10, 60, 35)
if __name__=='__main__' :
    app = QtWidgets.QApplication(sys.argv)
    fen = QuitButton()
    fen.show()
    app.exec_()
```

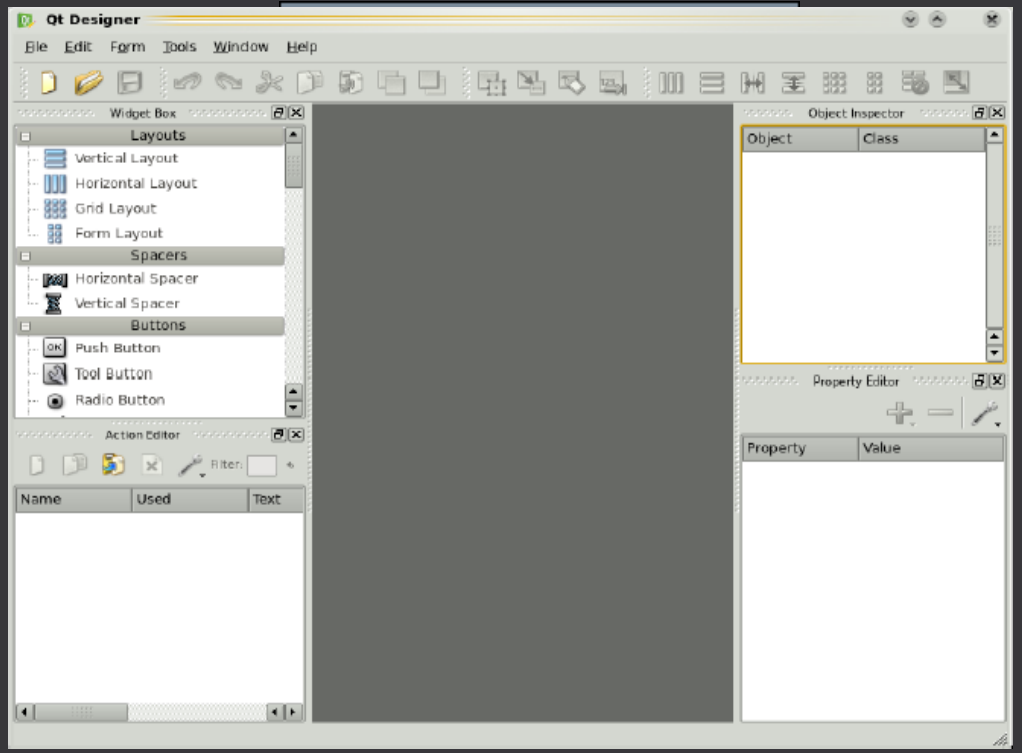


Sommaire

1. Création d'interface graphique (IHM)
   - Généralités
   - Composants graphiques de base
2. Présentation de Qt
   - Introduction
   - Les principaux modules
   - Documentation
3. IHM avec PyQt
   - Les modules de PyQt
   - Les widgets
   - Définition de classe graphique
   - Les fenêtres
   - Signaux et slots en Python

Les fenêtres

- La fenêtre principale est de type :
  - QMainWindow
  - Qdialog
- QMainWindow
  - Barre de menus : QMenuBar
  - Le conteneur (widget) central
    - setCentralWidget()
  - Barre d'outils : QToolBarre
  - Une barre d'état : QStatusBar
  - Les ancres (docks)

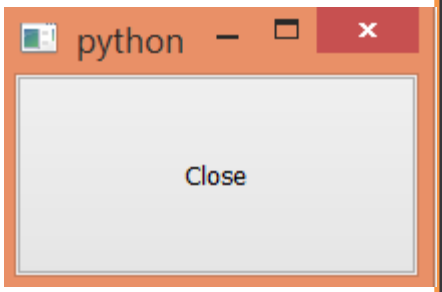


Les fenêtres

- QMainWindow

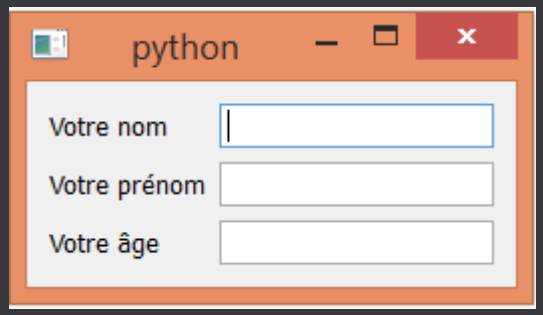
Exemple 1:

```
from PyQt5 import QtWidgets
import sys
class MyForm(QtWidget.QMainWindow):
    def __init__(self, parent =None):
        super().__init__(parent)
        the_button = QtWidget.QPushButton('Close', self)
        self.setCentralWidget(the_button)


if __name__=='__main__' :
    app = QApplication(sys.argv)
    fen = MyForm()
    fen.show()
    app.exec_()
```



Les fenêtres

- Exemple 2 :



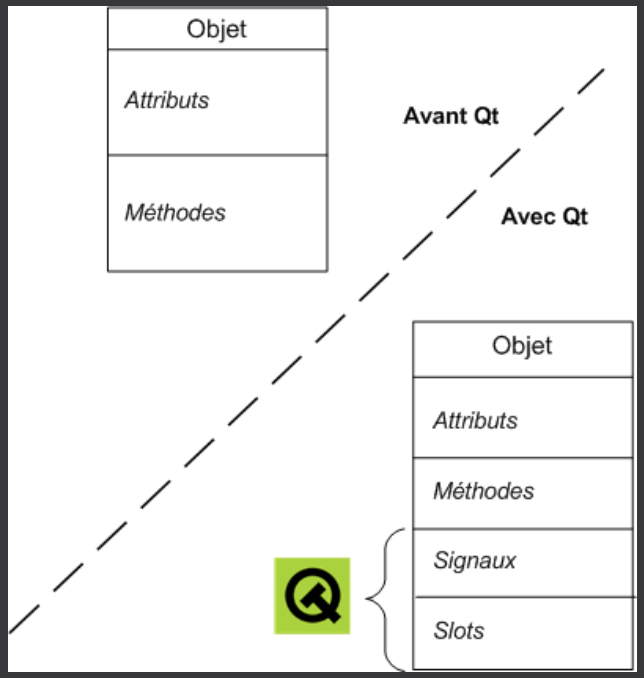

```
from PyQt5.QtWidgets import *
import sys
class MaFenetre(QMainWindow):
    def __init__(self, parent=None):
        super().__init__(parent)
        zoneCentrale = QWidget()
        nom = QLineEdit(self)
        prenom = QLineEdit(self)
        age = QLineEdit(self)

        layout = QFormLayout()
        layout.addRow("Votre nom", nom)
        layout.addRow("Votre prénom", prenom)
        layout.addRow("Votre âge", age)
        zoneCentrale.setLayout(layout)

        self.setCentralWidget(zoneCentrale)
if __name__=='__main__' :
    app = QApplication(sys.argv)
    fen = MaFenetre()
    fen.show()
    app.exec_()
```

Sommaire

1. Création d'interface graphique (IHM)
   - Généralités
   - Composants graphiques de base
2. Présentation de QT
   - Introduction
   - Les principaux modules
   - Documentation
3. IHM avec PyQt
   - Les modules de PyQt
   - Les widgets
   - Définition de classe graphique
   - Les fenêtres
   - Signaux et slots en Python

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.



Rappel : 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
     Connexion des signaux aux slots : gestion des évènements (QtCore)
  4. Appel de la méthode exec_() de l'application

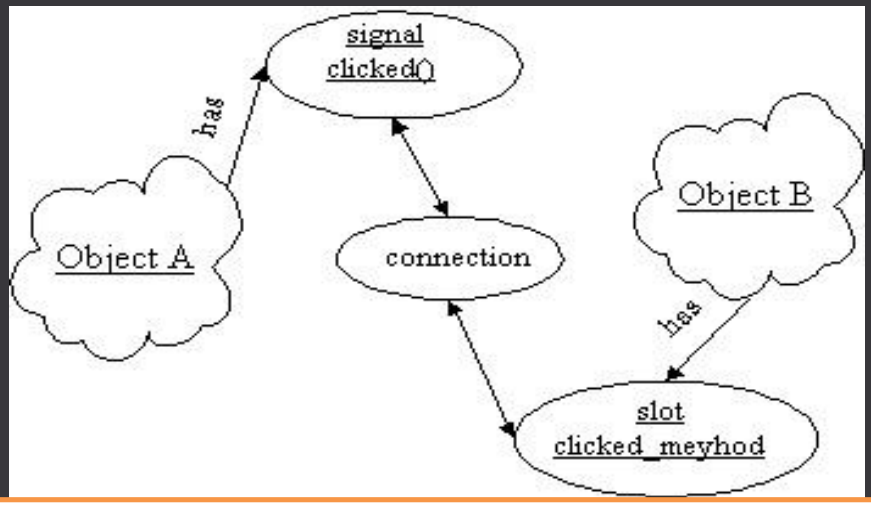
```
from PyQt5 import import QtWidgets
# 1 - Création d'une instance QApplication
app = QtWidgets.Application(args)
# 2 - Création de fenêtre
fen = QtWidgets.QMainWindow()
fen.setWindowTitle("Demo QMainWindow")
fen.resize(40,80)
fen.show()
# 2-a – Connexion des signaux aux slots
 # TODO
# 4 - Création de fenêtre
app.exec_()
```

Signaux et slots en Python

- Mécanisme utilisé dans QT pour assurer la communication entre les objets.

→ Module QtCore

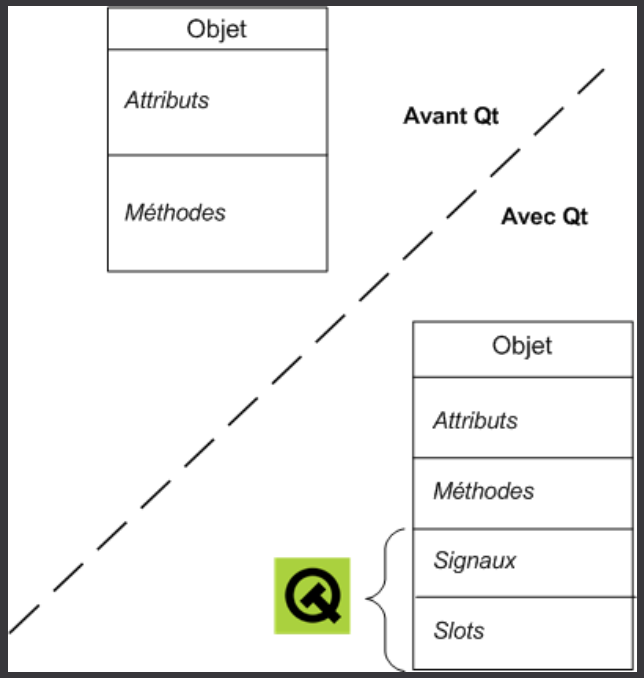
- Un signal : c'est un message envoyé par un widget lorsqu'un événement se produit. Exemple : on a cliqué sur un bouton.
- Un slot : c'est la fonction qui est appelée lorsqu'un évènement s'est produit. On dit que le signal appelle le slot. Concrètement, un slot est une méthode d'une classe.




```
ObjetA.signalName.connect(ObjetB.slotName)
```

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 :

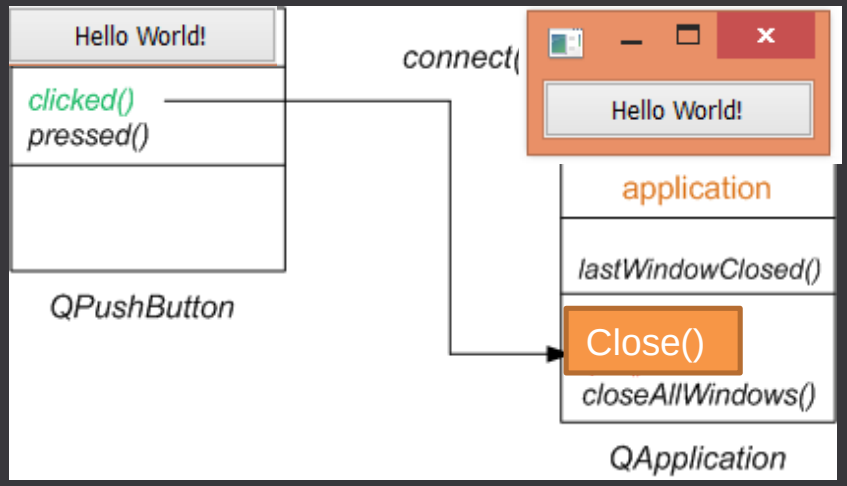
La connexion : la méthode connect() de l'objet QObject

Syntaxe :

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

Nous avons 4 éléments :

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 PyQt5.QtWidgets import *
from PyQt5.QtCore import *
import sys
class MyForm(QMainWindow):
  def __init__(self, parent=None):
      super().__init__(parent)
      self.the_button = QPushButton(‘Close‘, self)
      self.the_button.clicked(self.close)
      self.setCentralWidget(the_button)
```

Signaux et slots en Python

- Exemples :

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

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
import sys
class MyForm(QMainWindow):
  def __init__(self, parent=None):
    super().__init__(parent)
    self.the_button = QPushButton(‘Hello‘, self)
    self.the_button.clicked.connect(self.do)
    self.the_button.clicked.connect(self.close)
    self.setCentralWidget(self.the_button)
  def do(self):
    print(‘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 mêmes 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.

Définition un nouveau signal/slot

Signal

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

Exemple :

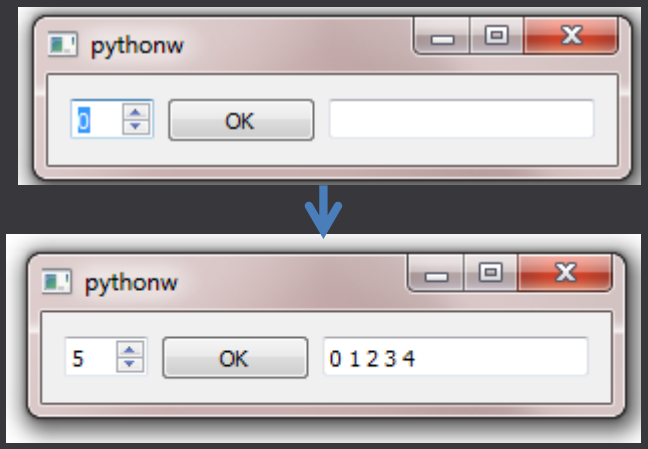
```
from PyQt5 import QtCore
mysignal = QtCore.pyqtSignal(list, int)
mysignal.emit([1,2,3,6],3)
# mysignal.emit([1,2,3,6], '3') ✂ error !!
```

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

```
mysignal.connect(slot/méthode)
```

Exemple

```
import sys
from PyQt5 import QtWidgets, QtCore
class MyWidget(QtWidgets.QWidget):
 mysignal = QtCore.pyqtSignal(list)
 def __init__(self, parent=None):
  super().__init__(parent)
  self.button = QtWidgets.QPushButton("OK", self)
  self.text=QtWidgets.QLineEdit(self)
  self.spin=QtWidgets.QSpinBox(self)
  self.grid=QtWidgets.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)
```



Préciser la valeur val de SpinBox, cliquer sur le ok, la liste (range(val)) est envoyée et affichée sur QlineEdit

```
def OnClicked(self):
 val=self.spin.value()
 self.mysignal.emit(list(range(val)))

def OnPrint(self, val):
  s= ''
  for el in val:
    s+=str(el)+' '
  self.text.setText(s)
```

QtDesigner

Projet informatique