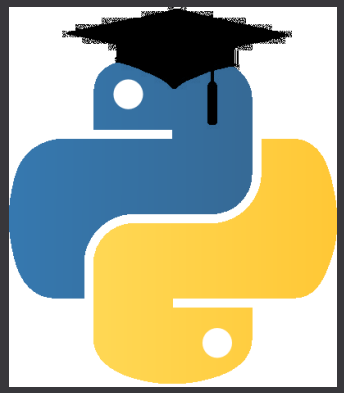


Python

PyQT pour l'IHM

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 a écrire

Solution :

- Utilisation des « boites à outils (toollkit graphique)»

→ Utilisent la programmation évènementielle

Généralités

Principe : La programmation d'IHM se décompose en 4 phases :

1. Assemblage des composants graphiques (widgets) constituant l'IHM
2. Ecriture des traitements à déclencher par les actions de l'utilisateur (modules de traitements)
3. Association des ces traitements aux actions de l'utilisateur (on associer le clique d'un bouton pour lancer un calcul)
4. Lancement de la boucle principale de la boite à outils

Remarque :

Ces phases sont plus au moins indépendantes selon la boite à outils utilisée

Composants graphiques de base

Type de composants (widgets) : Deux grandes catégories :

1. Les conteneurs : peuvent contenir d'autres composants
   - Les conteneurs de haut niveau (top level containers) : Ne peuvent être contenus dans d'autre conteneurs.

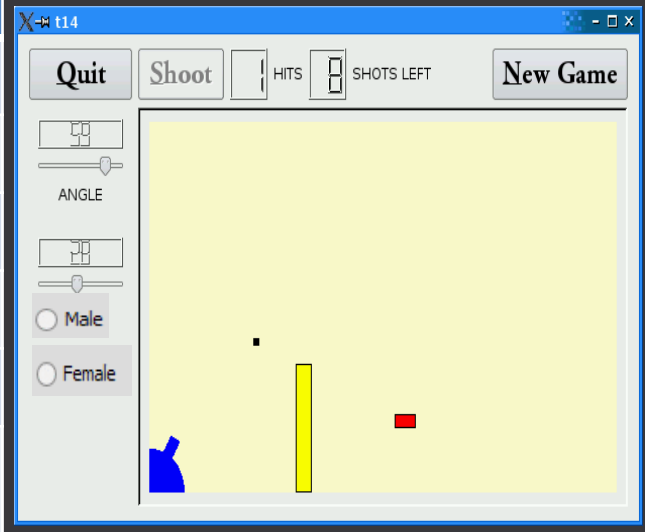
   Exemple : Fenêtre , boite de dialogue
2. Autres : ne peuvent contenir d'autre composants

   Exemple : Barre de défilement, Bouton de bascule (bouton radio)

Composants graphiques de base

- Disponibles dans toutes les boites à outils

Composants	Remarques
Fenêtre	conteneur de base
Bouton	composant muni d'un libellé -> cliquer
libellé	Texte affiché -> sans interaction
Champ texte	Zone permettant la saisi de texte
Bouton bascule	Composant avec un état binaire
Barre de défilement	Echelle permettant le défilement du contenu d'un conteneur
Zone de dessin	Conteneur (appelé canvas) destiné pour dessiner des primitives graphiques,,,



Composants graphiques de base

Disposition des composants (widgets)

- Pour un conteneur : on peut spécifier la disposition des composants
- Selon la boite à outils utilisée, la disposition peut être :
  - propre au conteneur
  - spécifiée par un objet externe associé au conteneur -> Layout

Exemple :

- Tkinter : *GridManager* permet de disposer les composants graphiques dans une grillebidimentielle
- GTK : Le conteneur *Hbox* dispose son contenu selon une bande horizontale

Composants graphiques de base

Quelques boites à outils graphiques

Qt	Windows, Mac and GNU/Linux	LGPL	C++, Python, Java, C#, Ruby, Ada, Pascal, Perl, PHP, Lua, Dao, Tcl, Lisp, D, Harbour
FLTK	UNIX/Linux (X11), Windows, and MacOS	LGPL 2	C++, Ruby, Tcl, Lua, Python
FOX	Linux, FreeBSD, MS-Window	LGPL	C++, Python, Ruby and Eiffel
GTK+	Windows, GNU/Linux and Unix, OSX and mobile devices	LGPL 2.1	C++ ,C# , Java , Python , JavaScript, Vala, Perl, Lua, Guile , Ruby , PHP, Ada , OCaml, D , Harbour
Ultimate++	Windows, Linux/FreeBSD, Mac OS		C++
MFC (Microsoft Foundation Classes)	Windows		Visual Studio obligatoire
Tkinter	Windows, Mac OS, GNU/Linux	Python Licence	python
VCF	Windows	BSD License	C++, Lua
WideStudio	Windows, Linux, FreeBSD, MacOSX	très permissive	C/C++, Java, Perl, Ruby, Python,Objective, Caml
wxWidgets	Windows, Mac OS, GNU/Linux, FreeBSD	LGPL +	C++, python, perl, java, lua, eiffel, C# (.NET), basic, ruby, javascript

Composants graphiques de base

Quelques boites 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 ;
- *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

  Enligne : ***http://doc.qt****.io/qt-4.8/*

  - Documentation de référence *: http://doc.qt.io/qt-4.8/modules.html*
  - *Via le navigateur d'aide QT Assistant : la commande >> assisstant*

  Codes sources des exemples de démos

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

Sommaire

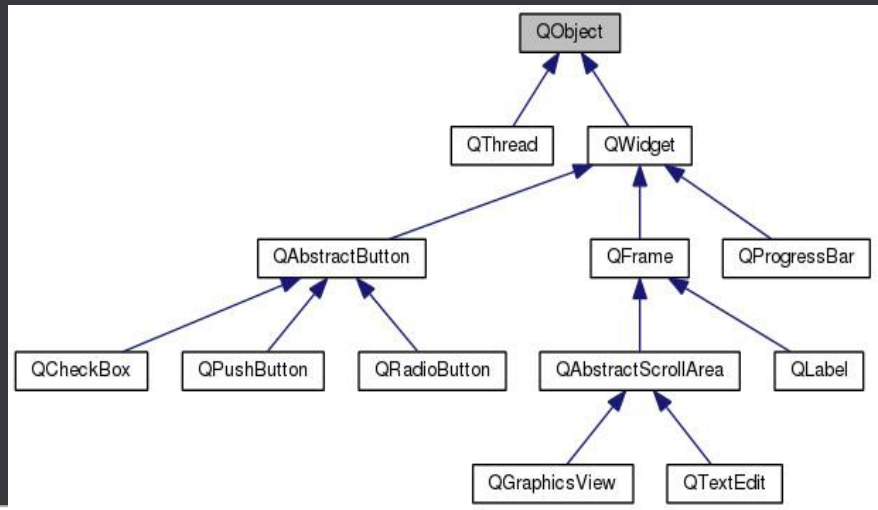
1. Création d'interface graphique (IHM)
   - Généralités
   - Composants graphiques de base
2. Présentation de QT
   - Introduction
   - Les principaux modules
   - Documentation
3. IHM avec PyQt
   - Les modules de PyQt
   - Les widgets
   - Définition de classe graphique
   - Les fenêtres
   - Signaux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

Modules de PyQt

- PyQt fournit un module Python PyQt4
- *PyQt4.QtCore* : pour les fonctionnalités non graphiques utilisées par les autres modules : mécanisme de support pour les signaux et les slots, …
- *PyQt4.QtGui* : pour les composants graphiques ;
- *PyQt4.QtNetwork*
- *PyQt4.QtOpenGL*
- *PyQt4.QtSql* ;
- *PyQt4.QtXml*
- *PyQt4.QtAssistant*
- *PyQt4.*QtWebKit
- *PyQt4.*QtTest

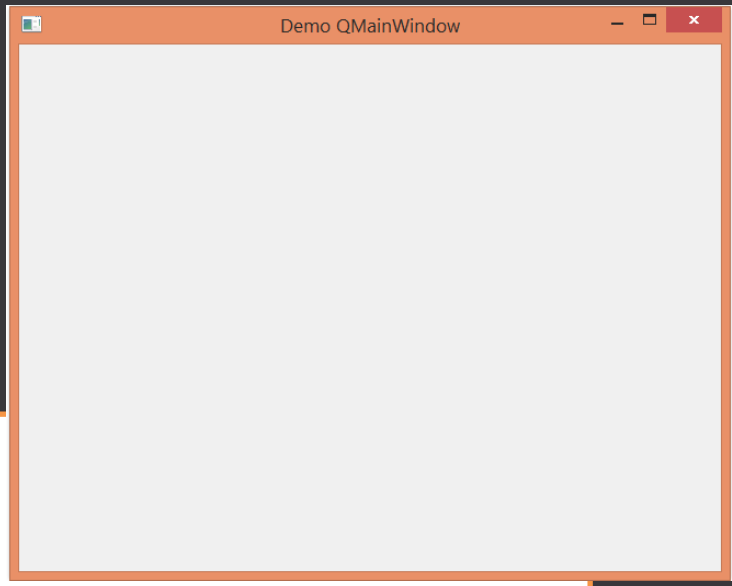
Les widgets

- Qwidget : la classe de base de tous les objets visibles dans l'IHM
- Les fenêtres : QMainWindow, QDialog
- Les conteneurs : Qframe, QGroupBox
- Les objets graphiques
  - Les boutons : QPushButton, QRadioButton
  - Les menus : QMenuBar, QPopMenu
  - Les barres d'outils : QToolBarre



Exemple 1 : widget simple

```
from PyQt4 import QtGui
import sys
app = QtGui.QApplication(sys.argv)
fen = QtGui.QWidget()
fen.setWindowTitle("Demo QMainWindow")
fen.resize(40,80)
fen.show()
app.exec_()
```

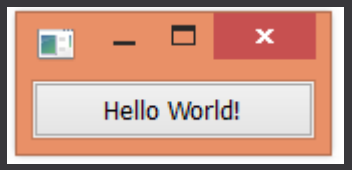



```
from PyQt4 import QtGui
import sys

def main(args) :
   #on doit disposer d'une instance de QApplication gérant l'ensemble des widgets
   app = QtGui.QApplication(args)
   #une fenetre
   fen = QtGui.QWidget()
   # titre de la fenetre
   fen.setWindowTitle("Demo QMainWindow")
   # taille de la fenetre
   fen.resize(40,80)
   #qu'on affiche
   fen.show()
   app.exec_()

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

Exemple 2 : création d'un bouton

Version simple :

```
from PyQt4 import QtGui
import sys
app = QtGui.QApplication(sys.argv)
hello = QtGui.QPushButton("Hello World!", None)
hello.show()
app.exec_()
```



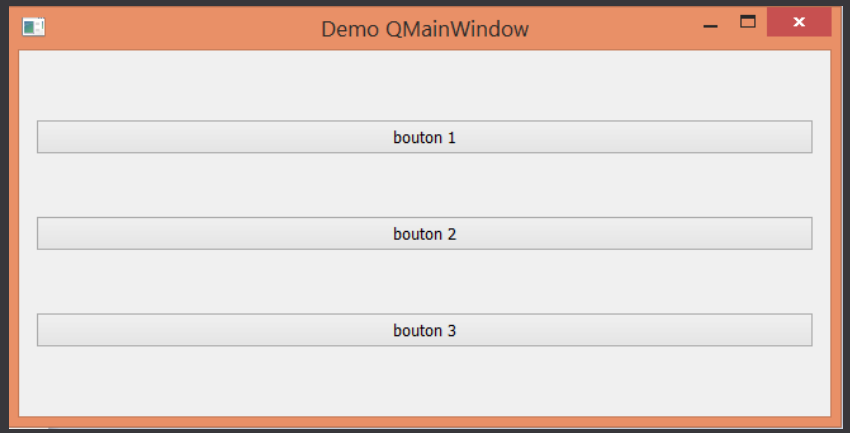
Version recommandée :

```
from PyQt4 import *
import sys

def main(args) :
    #on doit disposer d'une instance de QApplication gérant l'ensemble des widgets
    app = QApplication(args)
    #un nouveau bouton
    button = 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 widget 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 extensile
  - QGridLayout : placement des widgets en grille
    - addWidget(widget, row, col) : ajouter un widget à la position (row, col)
    - addMultiCellWidget(widget, fromRow, toRow, fromCol, toCol)

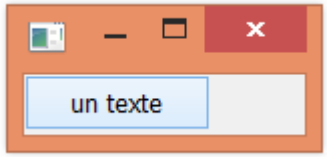
```
from PyQt4 import QtGui
import sys
app = QtGui.QApplication(sys.argv)
fen = QtGui.QWidget()
fen.setWindowTitle("Demo QMainWindow")
fen.resize(640,480)
layout=QtGui.QVBoxLayout()
bouton1= QtGui.QPushButton("bouton 1", fen)
bouton2= QtGui.QPushButton("bouton 2",fen)
bouton3= QtGui.QPushButton("bouton 3",fen)
layout.addWidget(bouton1)
layout.addWidget(bouton2)
layout.addWidget(bouton3)
fen.setLayout(layout)
fen.show()
app.exec_()
```



Définition d'une classe

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

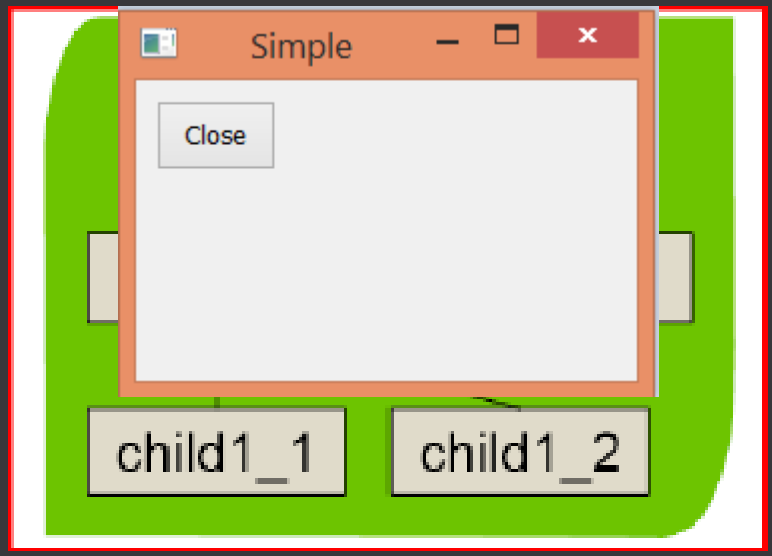
```
from PyQt4 import QtGui
import sys
class Fenetre(QtGui.QDialog):
   def __init__(self):
       super().__init__()
       self.setWindowTitle("Ma fenetre")
       self.bouton = QtGui.QPushButton('un texte', self)

if __name__=='__main__' :
    app = QtGui.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 supprimés, tous les objets ‘fils’ sont supprimés

```
import sys
from PyQt4 import QtGui
class QuitButton(QtGui.QWidget):
    def __init__(self, parent=None):
        QtGui.QWidget.__init__(self, parent)
        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Simple')
        quit = QtGui.QPushButton('Close', self)
        quit.setGeometry(10, 10, 60, 35)
if __name__=='__main__' :
    app = QtGui.QApplication(sys.argv)
    fen = QuitButton()
    fen.show()
    app.exec_()
```

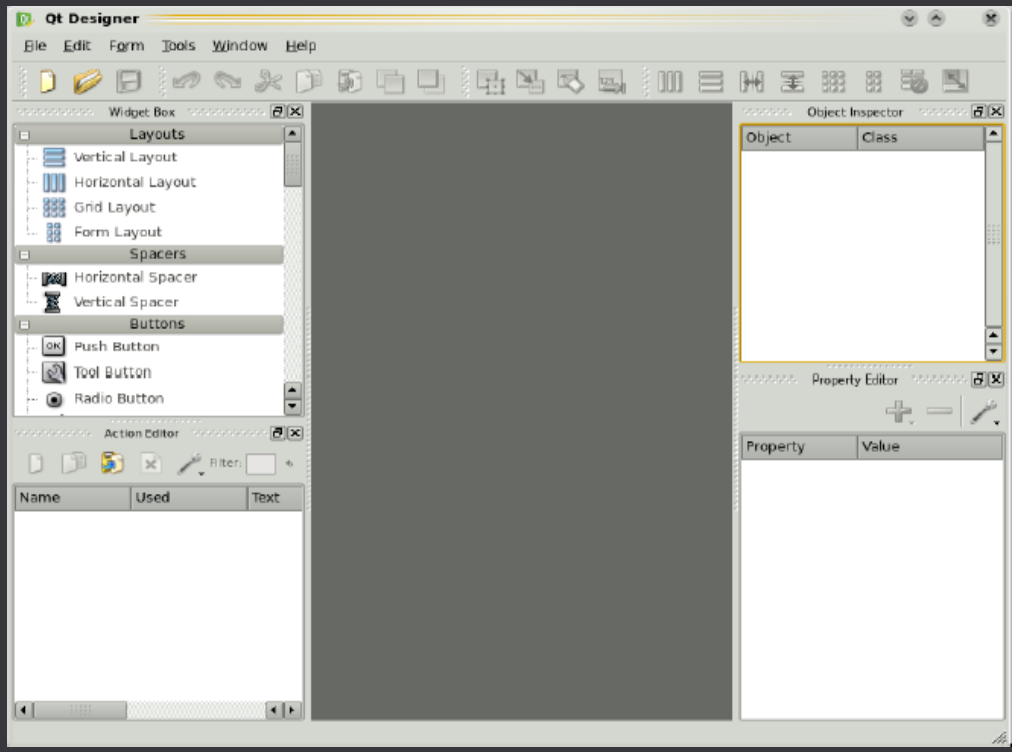


Sommaire

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)

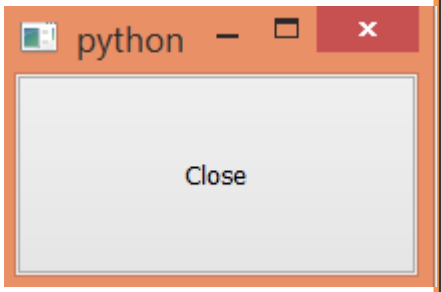


Les fenêtres

- QMainWindow

Exemple 1:

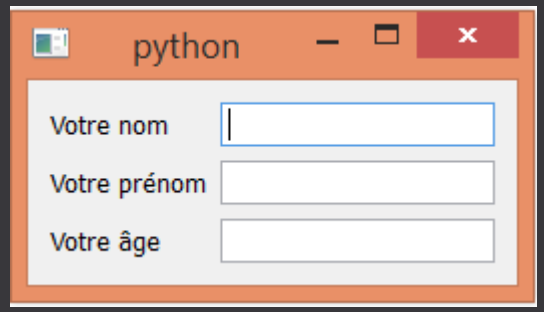
```
from PyQt4.QtGui import *
import sys
class MyForm(QMainWindow):
   def __init__(self):
       super(MyForm, self).__init__()
       the_button = QPushButton('Close', self)
       self.setCentralWidget(the_button)


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



Les fenêtres

- Exemple 2 :

```
from PyQt4.QtGui import *
import sys
class MaFenetre(QMainWindow):
    def __init__(self):
    QMainWindow.__init__(self)
    zoneCentrale = QWidget()
    nom = QLineEdit()
    prenom = QLineEdit()
    age = QLineEdit()
    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

Rappel :Structure d'un programme QT

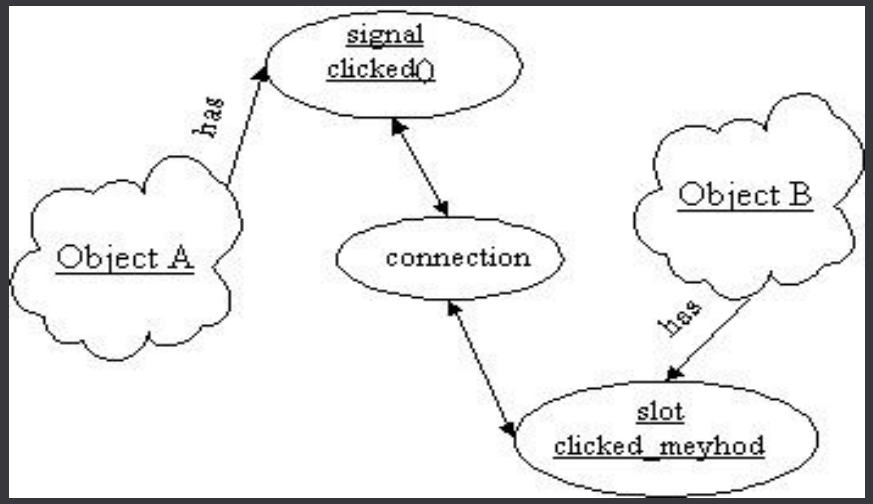
- Un programme PyQt typique contient les étapes suivante
  1. Création de l'instance de `QApplication`
  2. Création et affichage de la fenêtre principale et ses composants
  3. Connexion des signaux aux slots : gestion des évènements (QtCore)
  4. Appel de la méthode `exec__()` de l'application

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

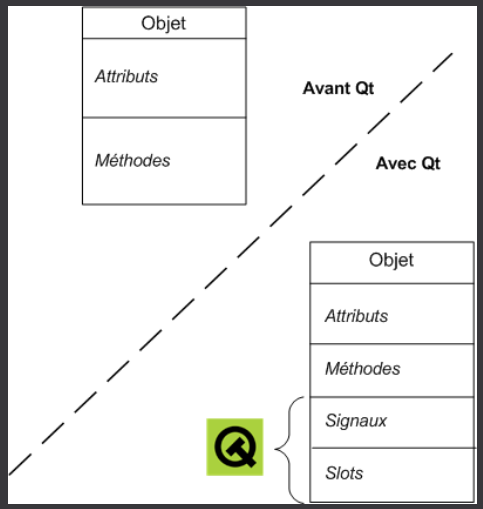
- Mécanisme utilisé dans QT pour assurer la communication entre les objets.
  - QTCore
- Un signal : c'est un message envoyé par un widget lorsqu'un évènement se produit. Exemple : on a cliqué sur un bouton.
- Un slot : c'est la fonction qui est appelée lorsqu'un évènement s'est produit. On dit que le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

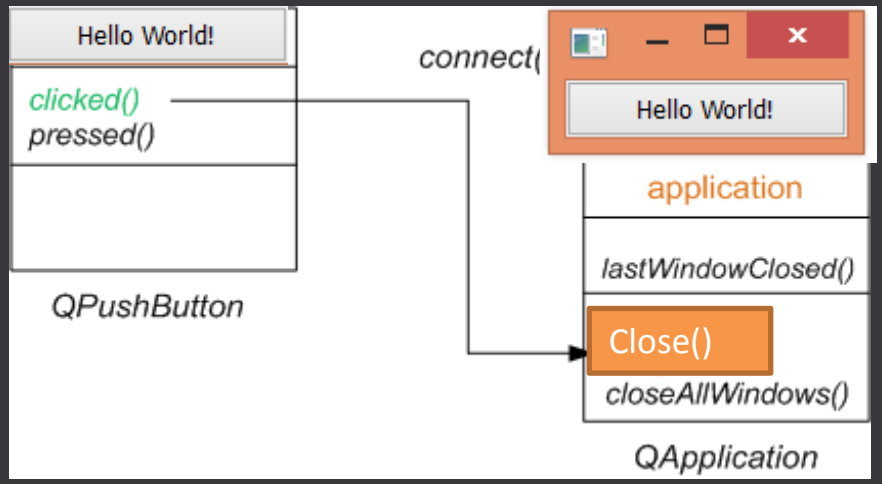
- La connexion : la méthode connect() de l'objet Qobject

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):
   QMainWindow.__init__(self,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 , 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.

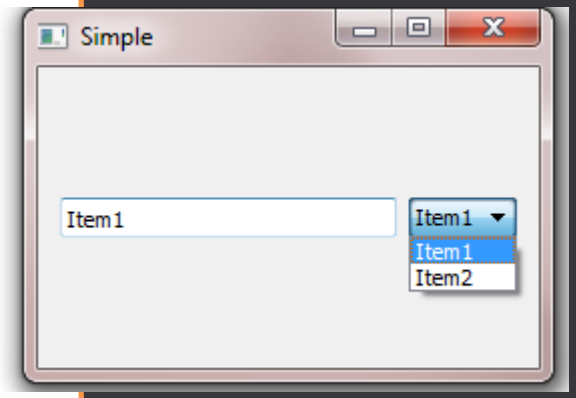
Exemple :

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 :

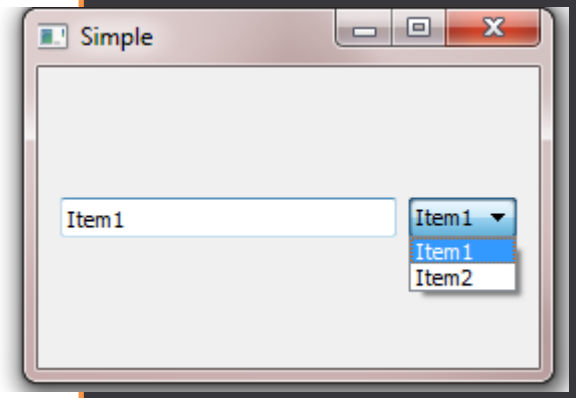
```
from PyQt4 import QtGui, QtCore
import sys
class Composant(QtGui.QWidget):
    def __init__(self,parent=None):
        QtGui.QWidget.__init__(self, parent)
        self.setWindowTitle('Simple')
        self.setGeometry(300,300,250,150)
        self.Line=QtGui.QLineEdit(' ',self)
        self.ComboBox=QtGui.QComboBox(self)
        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 :

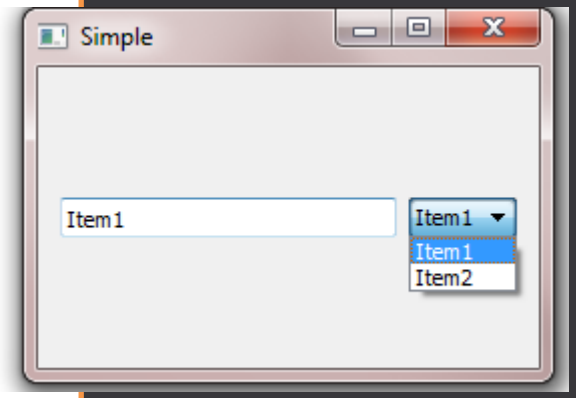
```
from PyQt4 import QtGui, QtCore
import sys
class Composant(QtGui.QWidget):
    def __init__(self,parent=None):
        QtGui.QWidget.__init__(self, parent)
        self.setWindowTitle('Simple')
        self.setGeometry(300,300,250,150)
        self.Line=QtGui.QLineEdit(' ',self)
        self.ComboBox=QtGui.QComboBox(self)
        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):
        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)'))
```



Définition signal/slot

Signal

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

Exemple :

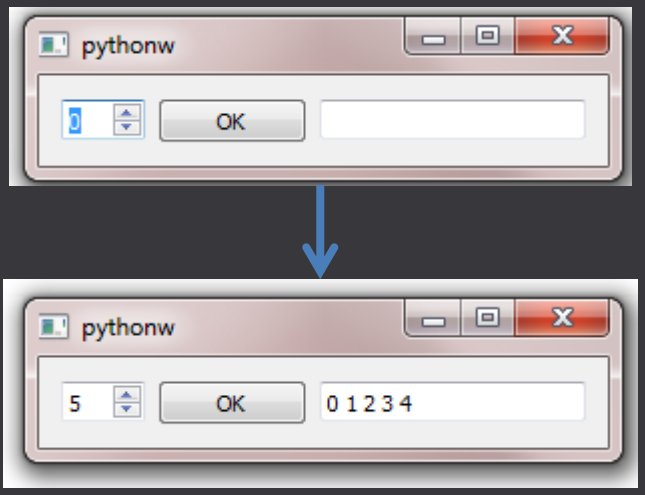
```
from PyQt4 import QtGui, QtCore
mysignal = QtCore.pyqtSignal(list, int)
mysignal.emit([1,2,3,6],3)
# error  :  mysignal.emit([1,2,3,6], '3')
```

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

```
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

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
4. QTDesigner

A vous de jouer

Projet informatique