

Python

Interface graphique avec Qt

D. DUBOIS

Diplôme Universitaire

CFI2L

Faculté d'Economie de Montpellier



1 PyQt

- Présentation
- Première interface graphique
- Classe QApplication
- Accesseurs et mutateurs
- Héritage

2 Principaux éléments graphiques

- Layout
- Widgets
- Windows
- Boîtes de dialogue

- 3 Gestion des événements
 - Fonctionnement
 - Signaux avec arguments
 - Principaux signaux

- 4 Qt Designer
 - Présentation
 - Manipulation

Qt

- Bibliothèque logicielle offrant des composants d'interface graphique (widgets)
- Développée par Trolltech, depuis 1991
- Développée en C++
- Sous licence LGPL ou propriétaire
- Utilisé par Skype, Google, Boeing, Adobe, la NASA ...
- Actuellement la version 5 (Qt5)

Autres bibliothèques graphiques

- framework .NET (windows). Utilisable avec C++ et c#
- Cocoa (Mac OS). Utilisable avec Objective C.
- GTK+ (Linux gnome).
- Aussi wxWidgets, FLTK

De nombreux modules

- QtCore : la boucle événementielle, le mécanisme des signaux/slots, les threads ...
- QtWidgets : éléments graphiques
- QtWebKit : pour implémenter un navigateur web
- QtMultimedia : pour le multimédia
- ...

Liste : <http://doc.qt.io/qt-5/modules-cpp.html>

- Multiplateforme
- Gratuit (licence LGPL)
- Simple d'utilisation
- Propose des outils comme Qt Designer pour la création des interfaces graphiques ou Qt Linguist pour l'internationalisation de l'application
- Utilise le look and feel natif du système

Beaucoup de documentation

- <http://doc.qt.io/>. La documentation est écrite pour du C++ mais les noms de classe, de fonction et d'attributs sont identiques en Python
- Utilisation fréquente de la documentation par classe : <http://doc.qt.io/qt-5/reference-overview.html> puis "All Classes"
 - ▶ Permet de trouver facilement la documentation d'une classe
 - ▶ Donne pour chaque classe sa classe mère, ses attributs, ses fonctions, ses signaux ...
 - ▶ Classement par ordre alphabétique, exclusion faite du "Q" car toutes les classes commencent par "Q"
- Liste des modules : <http://doc.qt.io/qt-5/modules-cpp.html>

Pour ce cours, utilisation principalement de QtCore, QtGui et QtWidgets

- QtCore est le coeur de Qt, tous les modules Qt s'appuient sur des classes de ce module (QTimer, QDate ...)
- QtGui contient des éléments utiles pour les applications graphiques (QImage, QColor, QFont ...)
- QtWidgets : contient les différents widgets utilisés pour les interfaces graphiques (QLineEdit, QRadioButton, QPushButton ...)

Interaction entre Qt et Python

- PyQt est un module qui permet de lier Python et Qt
- Permet de créer des interfaces graphiques Qt avec le langage Python
- Création des interfaces
 - ▶ Uniquement avec du code
 - ▶ Avec l'outil Qt Designer. Transformation ensuite du code Qt en code Python et appel des interfaces graphiques en Python.

```
import sys
from PyQt5 import QtCore, QtWidgets

if __name__ == '__main__':

    # création d'une application Qt
    app = QtWidgets.QApplication(sys.argv)

    # création d'un widget (fenêtre)
    w = QtWidgets.QWidget()
    w.setWindowTitle("Premier GUI")

    # création d'un layout
    layout = QtWidgets.QVBoxLayout()
    w.setLayout(layout)

    # création d'un label et ajout au layout
    label = QtWidgets.QLabel("Ceci est mon premier label")
    layout.addWidget(label)

    # création d'un bouton et ajout au layout
    b = QtWidgets.QPushButton("Ok")
    layout.addWidget(b, 0, QtCore.Qt.AlignRight)

    # affichage du widget
    w.show()

    # app.exec_() lance la "boucle" graphique
    # sys.exit() dit qu'il faut fermer l'application lorsque la boucle
    # s'arrête
    sys.exit(app.exec_())
```

Les éléments obligatoires

- Le shebang (env et coding)
- import du module sys
- import du module QtWidgets de PyQt5 :
`from PyQt5 import QtWidgets`
- Instanciation de la classe QApplication avec en argument sys.argv
- sys.argv contient le chemin du script lancé et les éventuelles options passées au lancement du script
- `app.exec_()` lance une boucle d'écoute d'événements
- `sys.exit(app.exec_())` termine le processus lorsque la boucle de l'application se termine

QApplication

- Doit être instanciée avant toute création d'objet graphique
- Une seule instance pour l'application
- Gère
 - ▶ l'ensemble des paramètres d'affichage de l'application
 - ▶ les événements système et les envoie aux widgets (clic souris, touches clavier ...)
 - ▶ l'ensemble des éléments graphiques de l'application (qui a le focus, position de chaque éléments ...)
 - ▶ l'internationalisation (langue système)

- accesseur : attribut()
- mutateur : setAttribut(valeur)

```
import sys
from PyQt5 import QtCore, QtWidgets

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)

    w = QtWidgets.QWidget()
    w.setWindowTitle("Deuxième GUI") # mutateur

    layout = QtWidgets.QVBoxLayout()
    w.setLayout(layout)

    label = QtWidgets.QLabel()
    label.setText("Ceci est mon second label") # mutateur
    layout.addWidget(label)

    label2 = QtWidgets.QLabel(w.windowTitle()) # accesseur
    layout.addWidget(label2)

    b = QtWidgets.QPushButton("Ok")
    layout.addWidget(b, 0, QtCore.Qt.AlignRight)

    w.show()

    sys.exit(app.exec_())
```

Pour créer et changer les attributs/méthodes d'un objet Qt il est plus simple de créer un objet qui hérite de cet objet

```
import sys
from PyQt5 import QtCore, QtWidgets

class MyWidget(QtWidgets.QWidget): # notre classe MyWidget hérite de QWidget
    def __init__(self):
        super().__init__() # instantiation de la classe mère

        # set the layout
        self.layout = QtWidgets.QVBoxLayout()
        self.setLayout(self.layout)

        # add a label
        self.label = QtWidgets.QLabel()
        self.label.setText("Ceci est mon troisième label")
        self.layout.addWidget(self.label)

        # add a button on the right
        self.button = QtWidgets.QPushButton("Ok")
        self.layout.addWidget(self.button, 0, QtCore.Qt.AlignRight)

        self.setWindowTitle("Troisième GUI")

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    wid = MyWidget()
    wid.show()
    sys.exit(app.exec_())
```

Permet d'agencer les éléments graphiques (widgets) dans la fenêtre

QVBoxLayout : vertical, les widgets sont placés les uns sous les autres

```
import sys
from PyQt5 import QtWidgets

class MyWidget(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()

        self.layout = QtWidgets.QVBoxLayout()
        self.setLayout(self.layout)

        self.label = QtWidgets.QLabel()
        self.label.setText("ici mon label 1")
        self.layout.addWidget(self.label)

        self.label1 = QtWidgets.QLabel()
        self.label1.setText("ici mon label 2")
        self.layout.addWidget(self.label1)

        self.setWindowTitle("Layout vertical")
        self.setMinimumWidth(200)

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    wid = MyWidget()
    wid.show()
    sys.exit(app.exec_())
```

QHBoxLayout : horizontal, les widgets sont placés les uns à côté des autres

```
class MyWidget(QWidgets.QWidget):
    def __init__(self):
        super().__init__()

        self.layout = QHBoxLayout()
        self.setLayout(self.layout)

        self.label = QLabel()
        self.label.setText("ici mon label 1")
        self.layout.addWidget(self.label)

        self.label1 = QLabel()
        self.label1.setText("ici mon label 2")
        self.layout.addWidget(self.label1)

        self.setWindowTitle("Layout vertical")
        self.setMinimumWidth(200)
```


QFormLayout : pour créer un formulaire a 2 colonnes : label | widget

```
class MyWidget(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()

        self.layout = QtWidgets.QVBoxLayout()
        self.setLayout(self.layout)

        self.form = QtWidgets.QFormLayout()
        self.layout.addLayout(self.form)

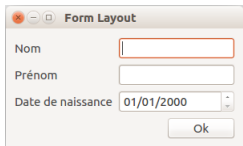
        self.lineEdit_nom = QtWidgets.QLineEdit()
        self.form.addRow(QtWidgets.QLabel("Nom"), self.lineEdit_nom)

        self.lineEdit_prenom = QtWidgets.QLineEdit()
        self.form.addRow(QtWidgets.QLabel("Prénom"), self.lineEdit_prenom)

        self.naissance_date = QtWidgets.QDateEdit()
        self.form.addRow(QtWidgets.QLabel("Date de naissance"),
                        self.naissance_date)

        self.button = QtWidgets.QPushButton("Ok")
        self.layout.addWidget(self.button, 0, QtCore.Qt.AlignRight)

        self.setWindowTitle("Form Layout")
        self.adjustSize()
```



QGridLayout : grille - Numérotation commence à 0, 0

```
class MyWidget(QWidget):
    def __init__(self):
        super().__init__()

        self.layout = QVBoxLayout()
        self.setLayout(self.layout)

        self.grid = QGridLayout()
        self.layout.addLayout(self.grid)

        self.grid.addWidget(QLabel("Nom"), 0, 0)
        self.lineEdit_nom = QLineEdit()
        self.grid.addWidget(self.lineEdit_nom, 0, 1)

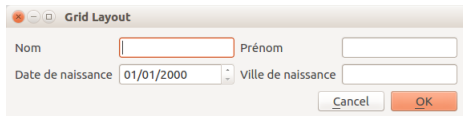
        self.grid.addWidget(QLabel("Prénom"), 0, 2)
        self.lineEdit_prenom = QLineEdit()
        self.grid.addWidget(self.lineEdit_prenom, 0, 3)

        self.grid.addWidget(QLabel("Date de naissance"), 1, 0)
        self.naissance_date = QDateEdit()
        self.grid.addWidget(self.naissance_date, 1, 1)

        self.grid.addWidget(QLabel("Ville de naissance"), 1, 2)
        self.lineEdit_naissance_ville = QLineEdit()
        self.grid.addWidget(self.lineEdit_naissance_ville, 1, 3)

        self.button = QDialogButtonBox(
            QDialogButtonBox.Cancel | QDialogButtonBox.Ok)
        self.layout.addWidget(self.button)

        self.setWindowTitle("Grid Layout")
        self.adjustSize()
```



Pour la documentation d'une classe : `http://doc.qt.io/qt-5/nom_classe`

- QLabel : pour afficher du texte (<http://doc.qt.io/qt-5/qlabel.html>)
- QLineEdit : pour saisie de texte sur une ligne (<http://doc.qt.io/qt-5/qlineedit.html>)
- QTextEdit : pour saisie/affichage de texte sur plusieurs lignes (<http://doc.qt.io/qt-5/qtextedit.html>)
- QDialogButtonBox : boutons annuler et ok (<http://doc.qt.io/qt-5/qdialogbuttonbox.html>)

label, lineEdit, textEdit

```
class MyWidget(QWidget):
    def __init__(self):
        super().__init__()

        self.layout = QVBoxLayout()
        self.setLayout(self.layout)

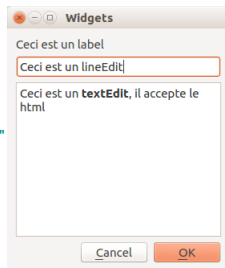
        self.label = QLabel()
        self.label.setText("Ceci est un label")
        self.layout.addWidget(self.label)

        self.lineEdit = QLineEdit()
        self.lineEdit.setText("Ceci est un lineEdit")
        self.layout.addWidget(self.lineEdit)

        self.textEdit = QTextEdit()
        self.textEdit.setHtml("<p>Ceci est un <b>textEdit</b>, il accepte le  
"le html")
        self.layout.addWidget(self.textEdit)

        self.button = QDialogButtonBox(
            QDialogButtonBox.Cancel | QDialogButtonBox.Ok)
        self.layout.addWidget(self.button)

        self.setWindowTitle("Widgets")
        self.adjustSize()
```



QRadioButton : boutons radios

<http://doc.qt.io/qt-5/qradiobutton.html>

```
class MyWidget(QWidget):
    def __init__(self):
        super().__init__()

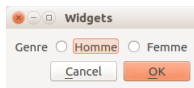
        # layout du widget
        self.layout = QVBoxLayout()
        self.setLayout(self.layout)

        # formulaire
        self.form = QFormLayout()
        self.layout.addLayout(self.form)

        self.radio_texts = ["Homme", "Femme"]
        self.radio_group = QButtonGroup()
        self.radio_layout = QHBoxLayout()
        for i, e in enumerate(self.radio_texts):
            rb = QRadioButton(e)
            self.radio_group.addButton(rb, i)
            self.radio_layout.addWidget(rb)
        self.radio_layout.addStretch(1) # afin que les boutons restent à gauche
        self.form.addRow(QLabel("Genre"), self.radio_layout)

        self._button = QDialogButtonBox(
            QDialogButtonBox.Cancel | QDialogButtonBox.Ok)
        self.layout.addWidget(self._button)

        self.setWindowTitle("Widgets")
        self.adjustSize()
```



QCheckbox : cases à cocher

<http://doc.qt.io/qt-5/qcheckbox.html>

```
class MyWidget(QWidget):
    def __init__(self):
        super().__init__()

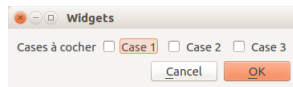
        self.layout = QVBoxLayout()
        self.setLayout(self.layout)

        self.form = QFormLayout()
        self.layout.addLayout(self.form)

        self.checkbox_texts = ["Case 1", "Case 2", "Case 3"]
        self.checkbox_list = []
        self.checkbox_layout = QHBoxLayout()
        for e in self.checkbox_texts:
            cb = QCheckBox(e)
            self.checkbox_list.append(cb)
            self.checkbox_layout.addWidget(cb)
        self.checkbox_layout.addStretch()
        self.form.addRow(QLabel("Cases à cocher"), self.checkbox_layout)

        self.button = QDialogButtonBox(
            QDialogButtonBox.Cancel | QDialogButtonBox.Ok)
        self.layout.addWidget(self.button)

        self.setWindowTitle("Widgets")
        self.adjustSize()
```



QComboBox : liste déroulante

<http://doc.qt.io/qt-5/qcombobox.html>

```
class MyWidget(QWidget):
    def __init__(self):
        super().__init__()

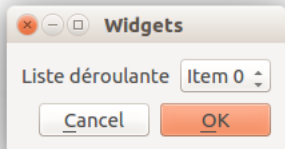
        self.layout = QVBoxLayout()
        self.setLayout(self.layout)

        self.form = QFormLayout()
        self.layout.addLayout(self.form)

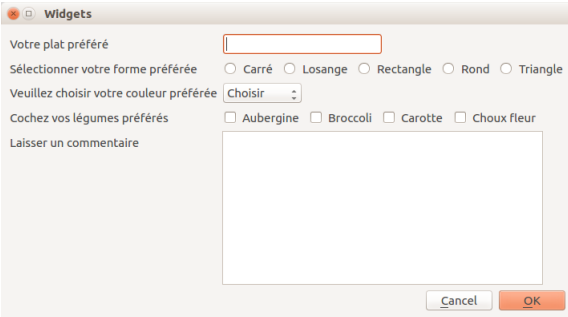
        self.combo_texts = ["Item 0", "Item 1", "Item 2", "Item 3"]
        self.combobox = QComboBox()
        self.combobox.addItem(self.combo_texts)
        self.form.addRow(QLabel("Liste déroulante"), self.combobox)

        self.button = QDialogButtonBox(
            QDialogButtonBox.Cancel | QDialogButtonBox.Ok)
        self.layout.addWidget(self.button)

        self.setWindowTitle("Widgets")
        self.adjustSize()
```



Dans un fichier widget.py créer le widget suivant



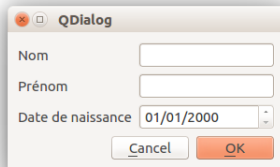
A screenshot of a Python Tkinter window titled "Widgets". The window contains the following elements:

- A text label "Votre plat préféré" followed by a single-line text entry field.
- A text label "Sélectionner votre forme préférée" followed by five radio button options: "Carré", "Losange", "Rectangle", "Rond", and "Triangle".
- A text label "Veuillez choisir votre couleur préférée" followed by a dropdown menu with the text "Choisir".
- A text label "Cochez vos légumes préférés" followed by four checkbox options: "Aubergine", "Broccoli", "Carotte", and "Choux fleur".
- A text label "Laisser un commentaire" followed by a multi-line text area.
- At the bottom right, there are two buttons: "Cancel" and "OK".

QDialog

- fenêtre modale (qui peut être attachée à une autre fenêtre)
- peut fournir une valeur de retour (souvent accept/reject)
- par défaut centrée sur sa fenêtre parente

```
class MyDialog(QDialog):  
    def __init__(self):  
        super().__init__()   
  
        self.layout = QVBoxLayout()  
        self.setLayout(self.layout)  
  
        self.form = QFormLayout()  
        self.layout.addLayout(self.form)  
  
        self.lineEdit_nom = QLineEdit()  
        self.form.addRow(QLabel("Nom"), self.lineEdit_nom)  
  
        self.lineEdit_prenom = QLineEdit()  
        self.form.addRow(QLabel("Prénom"), self.lineEdit_prenom)  
  
        self.naissance_date = QDateEdit()  
        self.form.addRow(QLabel("Date de naissance"), self.naissance_date)  
  
        self.button = QDialogButtonBox(  
            QDialogButtonBox.Cancel | QDialogButtonBox.Ok)  
        self.layout.addWidget(self.button)  
  
        self.setWindowTitle("QDialog")  
        self.adjustSize()
```



QMainWindow

- fenêtre principale d'une application
- possibilité d'ajouter des menus

```
class MyMainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

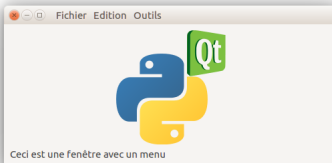
        # barre de menu
        self.menu_bar = QMenuBar()
        self.setMenuBar(self.menu_bar)
        self.menu_fichier = QMenu("Fichier")
        self.menu_bar.addMenu(self.menu_fichier)
        self.menu_edition = QMenu("Edition")
        self.menu_bar.addMenu(self.menu_edition)
        self.menu_outils = QMenu("Outils")
        self.menu_bar.addMenu(self.menu_outils)

        # il faut un widget en élément principal
        self.widget = QWidget()
        self.setCentralWidget(self.widget)
        self.layout = QVBoxLayout()
        self.widget.setLayout(self.layout)

        # chargement d'une image
        self.img_layout = QHBoxLayout()
        self.img = QPixmap()
        self.img.load("../img/pyqt.png")
        self.img_label = QLabel()
        self.img_label.setPixmap(self.img)
        self.img_layout.addWidget(self.img_label, 0, Qt.AlignCenter)
        self.layout.addLayout(self.img_layout)

        # un label
        self.label = QLabel("Ceci est une fenêtre avec un menu")
        self.layout.addWidget(self.label, 0, Qt.AlignLeft)

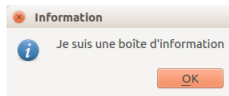
        self.setMinimumWidth(500)
        self.setWindowTitle("QMainWindow")
        self.adjustSize()
```



Information

```
import sys
from PyQt5 import QtWidgets

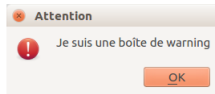
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    QtWidgets.QMessageBox.information(
        None, "Information", "Je suis une boîte d'information")
    sys.exit(app.exec_())
```



Warning

```
import sys
from PyQt5 import QtWidgets

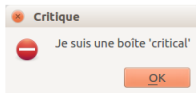
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    QtWidgets.QMessageBox.warning(
        None, "Attention", "Je suis une boîte de warning")
    sys.exit(app.exec_())
```



Critical

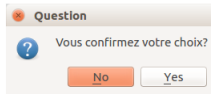
```
import sys
from PyQt5 import QtWidgets

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    QtWidgets.QMessageBox.critical(
        None, "Critique", "Je suis une boîte 'critical'")
    sys.exit(app.exec_())
```



Question

```
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    confirmation = QtWidgets.QMessageBox.question(
        None, "Question", "Vous confirmez votre choix?",
        QtWidgets.QMessageBox.No | QtWidgets.QMessageBox.Yes)
    sys.exit(app.exec_())
```

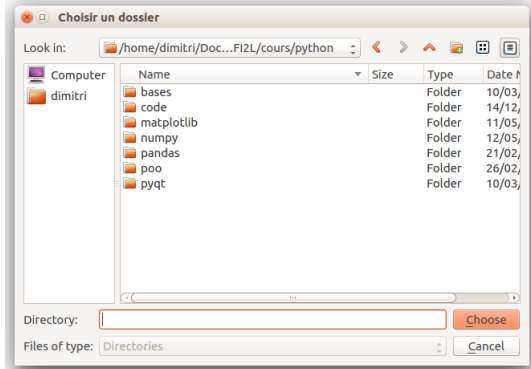


QFileDialog : Boîte de dialogue pour choisir un fichier ou un dossier

- `QtGui.QFileDialog.getExistingDirectory()` pour choisir un dossier
- `QtGui.QFileDialog.getOpenFileName()` pour choisir un fichier afin de l'ouvrir (mode lecture)
- `QtGui.QFileDialog.getSaveFileName()` pour choisir un fichier afin d'enregistrer (création ou ouverture mode écriture)

QFileDialog.getExistingDirectory

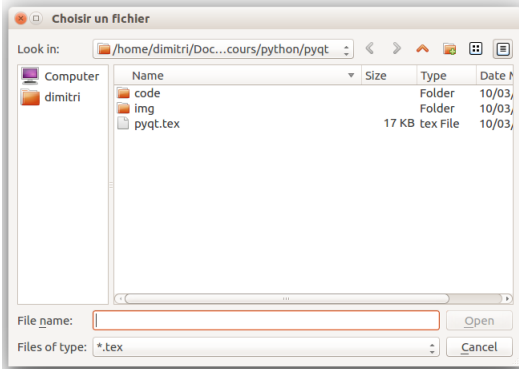
```
if __name__ == '__main__':  
    app = QtWidgets.QApplication(sys.argv)  
    dossier = QFileDialog.getExistingDirectory(  
        None, caption="Choisir un dossier",  
        directory="/home/dimitri/Documents/travail/CFI2L/cours/python")  
    if dossier:  
        QtWidgets.QMessageBox.information(  
            None, "Dossier", "Dossier choisi: {}".format(dossier))  
    sys.exit(app.exec_())
```



QFileDialog.getOpenFileName

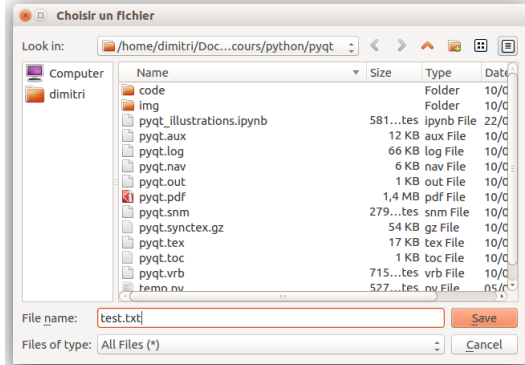
filter=*.extension permet de filter l'affichage

```
if __name__ == '__main__':  
    app = QtWidgets.QApplication(sys.argv)  
    fichier, _ = QtWidgets.QFileDialog.getOpenFileName(  
        None, caption="Choisir un fichier",  
        directory="/home/dimitri/Documents/travail/CFI2L/cours/python/pyqt",  
        filter="*.tex")  
    if fichier:  
        QtWidgets.QMessageBox.information(  
            None, "Dossier", "Fichier choisi: {}".format(fichier))  
    sys.exit(0)
```



QFileDialog.getSaveFileName

```
if __name__ == '__main__':  
    app = QtWidgets.QApplication(sys.argv)  
    fichier, _ = QtWidgets.QFileDialog.getSaveFileName(  
        None, caption="Choisir un fichier",  
        directory="/home/dimitri/Documents/travail/CFI2L/cours/python/pyqt")  
    if fichier:  
        QtWidgets.QMessageBox.information(  
            None, "Fichier", "Fichier choisi: {}".format(fichier))  
    sys.exit(0)
```



Signaux / Slots

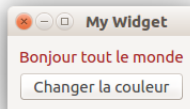
- Les objets émettent des signaux (clic sur un bouton, fermeture d'une fenêtre ...)
- Les signaux peuvent être connectés à des slots (fonctions appelées en réponse au signal)
- Ce mécanisme signaux/slots est très puissant et est une des caractéristiques de Qt
- Pour connecter un signal à un slot utilisation de la fonction connect
- Pour connaître les signaux émis par un objet Qt il faut regarder la documentation de Qt de l'objet dans la rubrique "Signals" (ou dans la classe mère de l'objet)

Exemple : Signaux émis par un pushbutton :

<http://doc.qt.io/qt-5/qabstractbutton.html#signals>

Click sur un bouton

```
class MyWidget(QWidget):  
    def __init__(self):  
        super().__init__()
  
        self.layout = QVBoxLayout()  
        self.setLayout(self.layout)
  
        self.label = QLabel("Bonjour tout le monde")  
        self.layout.addWidget(self.label)
  
        self.button = QPushButton("Changer la couleur")  
        self.button.clicked.connect(self.change_color)  
        self.layout.addWidget(self.button)
  
        self.setWindowTitle("My Widget")  
        self.adjustSize()
  
    def change_color(self):  
        col = choice(["blue", "yellow", "brown", "red", "pink", "black"])  
        self.label.setStyleSheet("color: {};".format(col))
```



Si le signal envoie un argument (cf. définition du signal) et que l'on veut l'utiliser alors il faut que le slot/la fonction le déclare

```
class MyWidget(QWidget):
    def __init__(self):
        super().__init__()

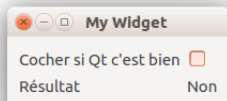
        form = QFormLayout()
        self.setLayout(form)

        self.checkbox = QCheckBox()
        form.addRow(QLabel("Cocher si Qt c'est bien"), self.checkbox)
        self.checkbox.stateChanged.connect(self.display_value)

        self.label = QLabel("Non")
        form.addRow(QLabel("Résultat"), self.label)

        self.setWindowTitle("My Widget")
        self.adjustSize()

    def display_value(self, state):
        if state == Qt.Checked:
            self.label.setText("Oui")
        elif state == Qt.Unchecked:
            self.label.setText("Non")
```



- QAction (ajoutée à QMenu) : triggered
- QCheckBox : stateChanged(int state)
- QDialogButtonBox : accepted (Ok), rejected (Cancel)
- QLineEdit : textChanged(const QString &text), editingFinished()
- QPushButton, QRadioButton : clicked
- QTextEdit : textChanged()
- QComboBox : currentIndexChanged(int index)
- ...

A partir de la fenêtre du slide [page 24](#)
(hériter de QDialog plutôt que de QWidget)

- lorsque click sur cancel ou sur la croix, fermeture de l'application après confirmation (Quitter l'application?)
- lorsque click sur ok, confirmation (Voulez-vous enregistrer les données saisies?)
- si oui, vérification que tous les champs sont remplis
- récupérer les textes, boutons sélectionnés etc.
- afficher les données dans une boîte de dialogue

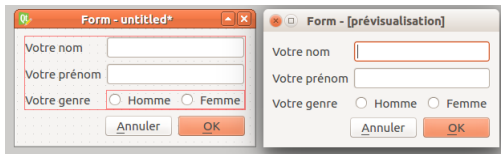
Aide : lorsque clic sur la croix, la fenêtre appelle la fonction reject. Il faut surcharger cette fonction.

- Outil fourni par Qt pour créer les interfaces graphiques en mode graphique
- Permet de créer et placer facilement les widgets, d'éditer/modifier leurs attributs, de définir des connections signaux/slots, de gérer l'ordre des focus ...
- Génère un fichier *.ui qu'il faut ensuite transformer en fichier python (*.py) grâce à la commande `pyuic5 fichier.ui -o fichier.py`

Documentation : <http://doc.qt.io/qt-5/qtdesigner-manual.html>

- Création d'une fenêtre
- Ajout de widgets (boutons, radiobutton, lineEdit, textEdit ...)
- Placement des widgets dans la fenêtre (Horizontal/vertical layout, horizontal/vertical spacer ...)
- Aperçu de la fenêtre
- Edition / modification des attributs/propriétés des widgets
- Edition des signaux/slots
- Edition de l'ordre des tabulations
- Conversion du fichier *.ui en *.py

Créer le widget suivant (à gauche en édition, à droite en prévisualisation)



- enregistrer le fichier
- le convertir en fichier *.py avec pyuic54

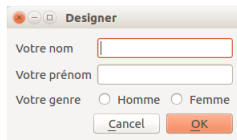
Utilisation du fichier *.py dans un module

- créer un module et faire un import du fichier py créé
- créer une classe qui hérite de QtGui.QDialog ou QtGui.QWidget selon
- créer dans cette classe un objet qui instancie soit Ui_Dialog soit Ui_Form selon (QDialog ou QWidget)
- appeler la méthode setupUi() de l'objet en passant self en argument
- paramétrer les attributs de l'objet et définir les méthodes

```
import sys
from PyQt5 import QtWidgets
import designer_1

class MyWidget(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()
        self.ui = designer_1.Ui_Form()
        self.ui.setupUi(self)
        self.setWindowTitle("Designer")

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    mywid = MyWidget()
    mywid.show()
    sys.exit(app.exec_())
```



- <http://pyqt.sourceforge.net/Docs/PyQt4/>
- <http://zetcode.com/gui/pyqt4/>
- Openclassroom
- <http://www.rkblog.rk.edu.pl/w/p/introduction-pyqt4/>