Programmation d'interfaces graphiques

Interfaces graphiques avec Python

- Il existe de nombreux « toolkits » graphiques, disponibles sur tous les systèmes (plus ou moins facilement...) : *Tk*, *Gtk*, *Qt*, *wxWidgets*.
- Ces toolkits étant généralement écrits en C ou en C++, les différents langages utilisent des « wrappers » afin d'accéder à leurs API. Pour les utiliser avec Python, il faut que les bibliothèques correspondantes soient installées sur votre système.
- Python est généralement fourni avec le module *tkinter* (*Idle* est une application Python/tkinter). Il est assez simple à programmer mais son look est un peu « daté ».
- GTK a été initialement conçu pour le développement du logiciel *The Gimp*, il a évolué en GTK2 et sa version actuelle, GTK3, est le toolkit utilisé pour l'environnement Gnome3. Pour l'utiliser avec Python, il faut installer *pyGTK* (pour GTK2) ou *pyGObject* (pour GTK3). Pour faciliter la création des interfaces GTK, on dispose de l'outil *Glade*.

Interfaces graphiques avec Python

- QT de Trolltech (devenu Qt Software depuis son rachat par Nokia) est la bibliothèque sur laquelle repose l'environnement KDE. Il a évolué en QT4, puis QT5 (avec quelques problèmes de compatibilité). Pour l'utiliser avec Python, il faut installer le paquetage *PyQt5* de Riverbank (utilisation libre pour les projets GPL, sinon payant) ou *PySide* (libre). L'outil *QT Designer*, fourni avec la distribution QT permet de créer des interfaces graphiques et l'outil *pyuic5* permet de les convertir en code Python.
- Pour ce cours, nous avons choisi *PyQt5*.

Python et QT5

- Le développement d'une application QT5 en Python s'effectue en plusieurs étapes :
  1. Création de l'interface de l'application avec *QT Designer* afin d'obtenir un fichier *.ui* au format XML (donc exploitable par tout langage disposant d'un wrapper QT).
  2. Conversion de ce fichier *.ui* en code source Python par la commande *pyuic5* : le fichier obtenu contient une classe Python décrivant l'interface graphique de l'application.
  3. Écriture des classes « métier » de l'application, dont l'une (la classe « principale ») héritera de la classe de l'interface graphique.
  4. Écriture d'un programme principal qui consistera à instancier cette classe et à créer la boucle d'événement du toolkit.

Remarques :

- La pratique consistant à mélanger le code de l'interface et le code métier est à proscrire (c'est malheureusement assez fréquent dans les solutions pourries qu'on trouve sur les forums pour apprentis programmeurs...)
- Bien que le fichier *.ui* ne soit plus nécessaire après la création de la classe Python, conservez-le si vous voulez ensuite modifier l'interface (il faudra alors regénérer la classe Python correspondante).

Principe général de l'application

- Python autorisant l'héritage multiple, la classe principale peut dériver à la fois de la classe QT correspondant au type de la fenêtre de notre application (*QWidget*, par exemple) et de la classe de notre interface, créée par *pyuic5*.
- Le constructeur de cette classe appelera le constructeur du widget principal, mettra en place l'interface graphique puis connectera les différents widgets pour qu'ils répondent aux actions de l'utilisateur.
- Pour illustrer cette démarche, nous supposerons que nous avons créé une classe Python nommée *Ui_ToutEnMaj* (QT Designer crée cette classe en rajoutant *Ui_* devant le nom que l'on a donné à notre widget principal). Nous avons choisi *QWidget* comme widget de base dans QT Designer et nous avons appelé ce widget *ToutEnMaj*.
- Cette interface contient un label (appelé *label*), un champ de saisie (appelé *lineEdit*) et un bouton (appelé *pushButton*).
- Nous voulons que le contenu du champ de saisie passe de minuscules en majuscules (et inversement) à chaque clic du bouton.

Écriture de la classe principale

- La classe principale doit importer certains modules de *PyQt5*, ainsi que notre classe *Ui_ToutEnMaj* que nous avons sauvegardée dans le fichier *toutenmaj.py* :

```
from PyQt5 import QtCore
from PyQt5.QtWidgets import QApplication, QWidget
from toutenmaj import Ui_ToutEnMaj
```

- Elle doit dériver de *QWidget* et de *Ui_ToutEnMaj* et son constructeur doit appeler celui de *QWidget* :

```
class MajQt(QWidget, Ui_ToutEnMaj):
    def __init__(self, parent=None):
        QWidget.__init__(self, parent)
```

- Puis, le constructeur doit mettre en place notre interface graphique par un appel à la méthode *setupUi* de *Ui_ToutEnMaj* :

```
        self.setupUi(self)
```

- Enfin, il connecte le clic du bouton à une méthode *on_clic* que nous écrirons ensuite :

```
        self.pushButton.clicked.connect(self.on_clic)
```

Application principale (suite)

- Il reste à écrire la méthode *on_clic* :

```
def on_clic(self):
    self.lineEdit.setText(self.lineEdit.text().swapcase())
```

- Et le programme principal de l'application, qui met en place l'environnement QT, crée une instance de notre classe et lance l'application :

```
if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)
    win = MajQt()
    win.show()
    sys.exit(app.exec_())
```

Slots automatiques

- Dans la terminologie QT, les widgets *émettent* des *signaux*. Par exemple, un `QPushButton` peut émettre les signaux *clicked*, *pressed*, *released*, etc. (voir l'éditeur de signaux dans QT Designer).
- Les signaux sont traités par des *slots* qui sont représentés par des fonctions qui exécuteront leur code lorsque le signal sera reçu. En PyQT, on *connecte* à un slot le signal d'un widget à l'aide de la méthode *connect* :

```
self.pushButton.clicked.connect(self.on_clic)
```

- À partir de QT4, cette connection peut être automatique : il suffit qu'une méthode soit décorée par *@QtCore.pyqtSlot()* et que son nom soit de la forme *on_widget_signal(self)* pour que cette méthode soit considérée comme le slot du signal indiqué pour le widget indiqué.

Slots automatiques

La classe principale de l'exemple précédent devient donc :

```
from PyQt5 import QtCore
from PyQt5.QtWidgets import QApplication, QWidget
from toutenmaj import Ui_ToutEnMaj


class MajQt(QWidget, Ui_ToutEnMaj):
    def __init__(self, parent=None):
        QWidget.__init__(self, parent)
        self.setupUi(self)
        self.pushButton.setToolTip("Bascule la casse")

    @QtCore.pyqtSlot()
    def on_pushButton_clicked(self):
        self.lineEdit.setText(self.lineEdit.text().swapcase())

if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)
    win = MajQt()
    win.show()
    sys.exit(app.exec_())
```

Gestion des menus

- La gestion des menus avec QT est assez simple : le widget principal doit être un *QMainWindow* (et non un *QForm)*.
- Essentiellement, un *QMainWindow* possède une barre de menu (un *QMenuBar*), dans laquelle on place des *QMenu*.
- Chaque *QMenu* comporte des *QAction* qui représentent soit des options du menu, soit des séparateurs.
- Chaque *Qaction* émet le signal *triggered* lorsqu'il est sélectionné, *hovered* lorsque le curseur s'attarde sur lui, etc.
- Un *QMainMenu* possède également un *QStatusBar* qui permet de gérer une barre de statut située en bas de la fenêtre.

Exemple de gestion de menus

```
from PyQt5 import QtCore, QtGui
from PyQt5.QtWidgets import QApplication, QMainWindow, QFileDialog
import sys

from FichiersUI import Ui_MainWindow

class OpenFileQT(QMainWindow, Ui_MainWindow):
    def __init__(self, parent=None):
        QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.actionQuitter.triggered.connect(QtGui.qApp.quit)

    @QtCore.pyqtSlot()
    def on_actionOuvir_triggered(self):
        nomfic, _ = QFileDialog.getOpenFileName(
                    self,
                    "Ouvrir un fichier",
                    QtCore.QDir.homePath(),
                    "Fichiers images (*.png *.jpg);; Tous les fichiers (*)"
                    )
        if nomfic:
            self.label.setPixmap(QtGui.QPixmap(nomfic))

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

Pour en savoir plus sur PyQT

- La documentation de PyQT : *http ://pyqt.sourceforge.net/Docs/PyQt5/*
- Des questions et des réponses sur StackOverflow : *http ://stackoverflow.com/search ?q=pyqt5*
- La documentation officielle de QT : *http ://qt-project.org/doc/*
- Attention aux « solutions » trouvées sur les forums ! Vérifiez la version de Python (3.x et non 2.x), vérifiez la version de PyQT, vérifiez qu'elle utilise bien les dernières évolutions (notamment pour la gestion des signaux et des slots), vérifiez qu'elles ne mélangent pas le code de l'UI et le code métier.