Programmation orientée objet avec Python

Marie-Dominique Van Damme, ENSG

Cycle d'ingénieur - Master mention géomatique
mars 2024

Source de la section d'introduction issue du cours : Programmation orientée objet avec python, M-D. Van Damme, Y. Méneroux, 2020-2021

Table des matières

Plan

Plan

Interface graphique

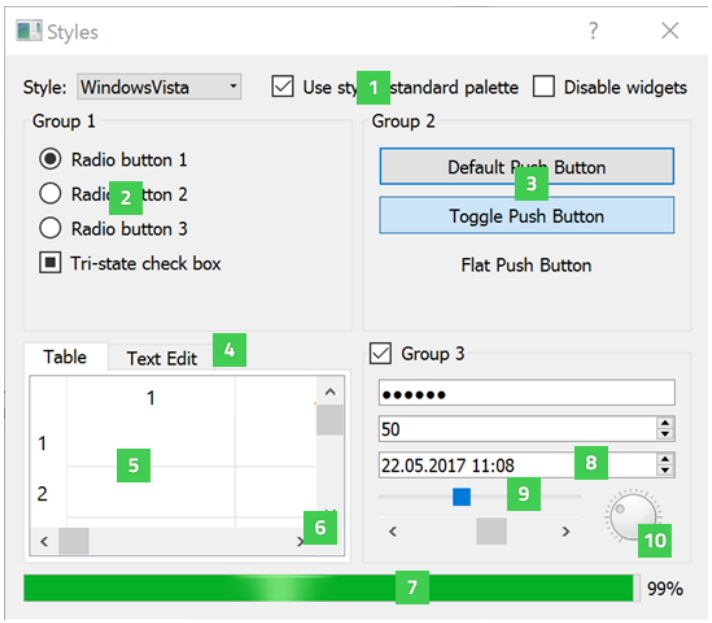
(en anglais **GUI** pour graphical user interface)

Une application GUI est composée de plusieurs composants qui communiquent entre eux. Ils peuvent être graphiques, c'est à dire visibles à l'écran, ou non graphiques.

Parmi les composants graphiques d'une application donnée, on aura en général une fenêtre principale et plusieurs fenêtres secondaires (typiquement des fenêtres de dialogue) qui apparaîtront au gré des besoins.

Widgets

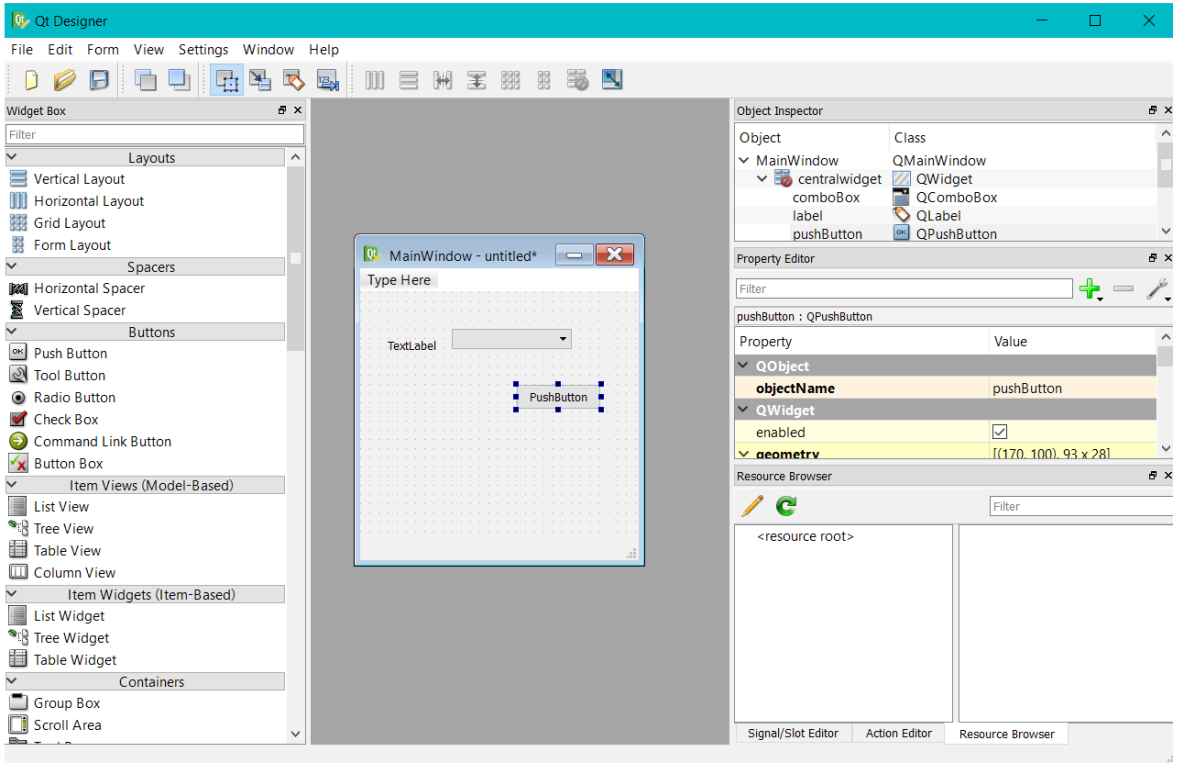
Le programmeur a pour rôle de créer ces fenêtres, selon un assemblage de divers composants graphiques appelés **widgets**. Ces widgets sont fournis par la bibliothèque graphique.



Certaines fenêtres sont prêtes à l'emploi. Par exemple, une fenêtre de dialogue pour sélectionner un fichier ou pour inviter l'utilisateur à répondre à une question oui/non.

QtDesigner

Qt met à disposition des développeurs une application puissante pour dessiner les fenêtres sans avoir à saisir une ligne de code : Qt Designer.



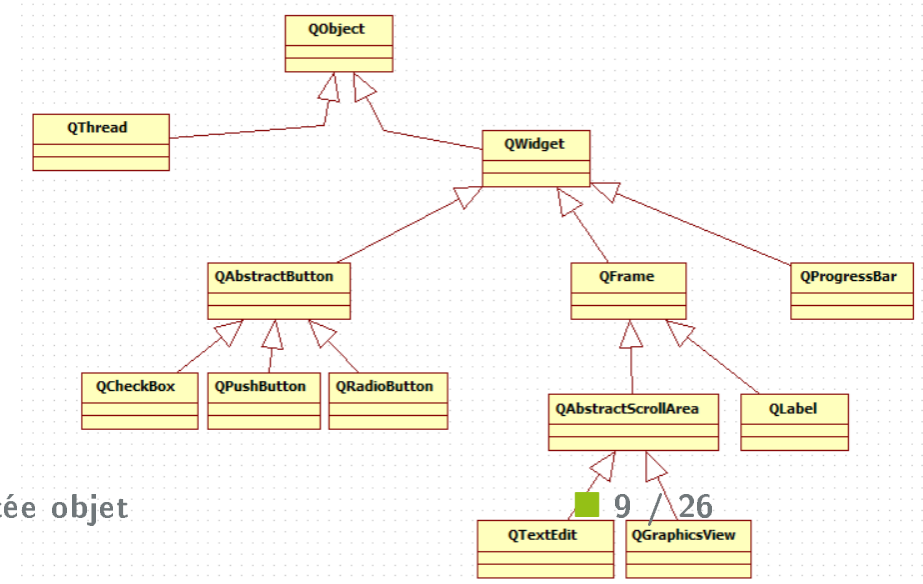
Programmation orientée objet

La programmation d'une interface graphique se base sur le paradigme de l'orientation objet (OO). Chaque élément de l'interface utilisateur est un objet : il a ses attributs et ses méthodes.

PyQt utilise la technique de l'héritage de façon poussée : les classes à utiliser par le programmeur d'application font partie d'une hiérarchie de classes ; aussi les méthodes accessibles dans une classe donnée peuvent être documentées dans une classe parente ou ancêtre.

Diagramme de classes des composants graphiques

La classe abstraite *QWidget* intègre toute la gestion du positionnement et du dimensionnement d'un élément. Les classes des éléments affichés, comme le champ texte (*QTextEdit*), le bouton (QPushButton), héritent de la classe *QWidget* et donc comment elle doit dessiner quelque chose dans une zone rectangulaire sur l'écran. Ce qu'il faut dessiner exactement est défini dans les classes enfants



Plan

Programmation évènementielle

Une application est beaucoup plus qu'un assemblage de composants graphiques sans interactivité, sinon ce ne serait qu'une "nature morte". Selon les actions effectuées par l'utilisateur, il faudra exécuter certains traitements qui peuvent influer les composants affichés. On parle de programmation évènementielle pour ce type d'application.

Une application de type interface utilisateur, après initialisation, se met en attente et réagit aux évènements qui se présentent. On parle de boucle d'évènements pour désigner la boucle principale qui réceptionne les évènements et les dirige vers les fonctions de traitement adéquates.

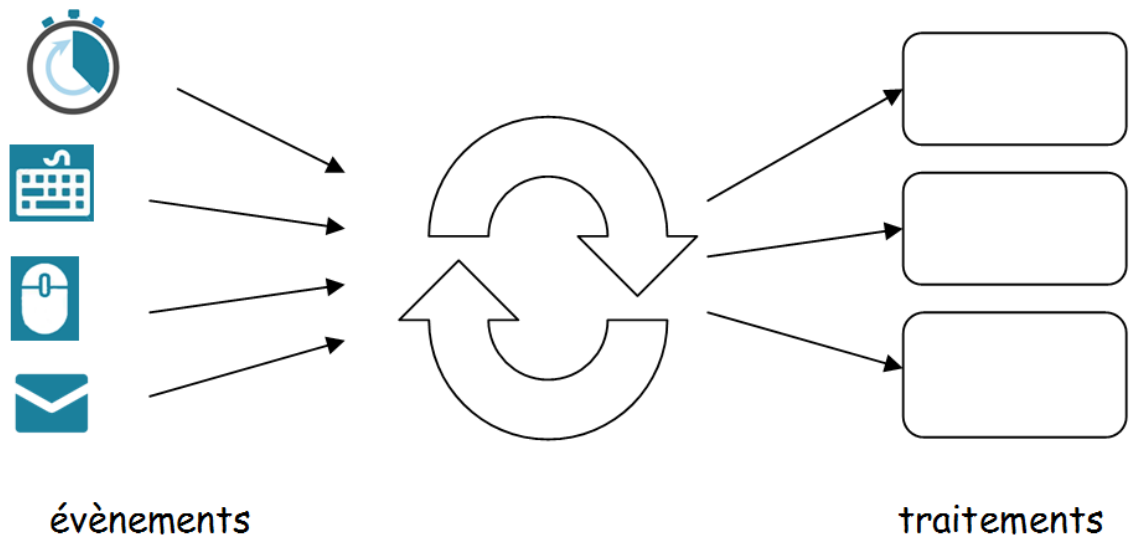
Évènements

Quels sont ces évènements ? En fait il y en a beaucoup : clics et déplacements de la souris, enfoncement/relâchement d'une touche au clavier, etc.

Outre ces évènements liés aux actions de l'utilisateur, d'autres sont d'ordre programmatique : fin d'une tâche, réception d'un message, échéance d'un intervalle de temps, etc. C'est par leur biais que l'affichage peut se mettre à jour, même lorsque l'utilisateur ne fait rien : par exemple une barre d'avancement qui évolue, une petite enveloppe qui apparaît lorsqu'un nouvel email est arrivé.

Boucle d'évènements

La boucle d'évènements vérifie si de nouveaux évènements sont apparus ; le cas échéant, elle appelle la fonction de traitement correspondante.



Évènements

Fort heureusement, les composants de PyQt gèrent par eux-mêmes la majorité des évènements et réagissent de manière adaptée, soulageant le programmeur d'une multitude d'activités de bas niveau : redimensionnement d'une fenêtre, affichage et parcours de menus, changement de couleur quand le curseur passe au-dessus d'un élément, infobulles, etc. Le programmeur peut dès lors se concentrer sur les évènements spécifiques à l'application et les traitements qui lui sont associés.

Signal et slot

Lorsqu'un évènement se produit un **signal** est émis, identifiant l'évènement, le composant où il se produit et d'éventuels paramètres associés.

Le programmeur peut connecter ce signal à une méthode spéciale appelée **slot**. Cette méthode s'exécutera alors à chaque fois que le signal est émis.

Un signal ne doit pas forcément être connecté à un slot (typiquement, si le traitement de PyQt est suffisant)

Si nécessaire, le même signal peut être connecté à plusieurs slots, qui s'exécuteront tous à l'émission du signal.

Plan

Champ texte : *QLineEdit*

```
e1 = QLineEdit()
e1.setMaxLength(4)

e1.setText('Texte par défaut')
print (e1.text())
```

Signal : *textChanged*. Exemple :

```
e1.textChanged.connect(fonctionTextChanged)

def fonctionTextChanged():
    ....
```

Bouton radio : *QRadioButton*

```
b1 = QRadioButton("Button1")
b1.setChecked(True)
if cb.isChecked():
    print ("case cochée")
```

Signaux : *click*, *toggled*. Exemple :

```
b1.toggled.connect(fonctionRadioBtn)

def fonctionRadioBtn():
    ....
```

Liste déroulante : *QComboBox*

```
cb = QComboBox()

cb.addItem("C")
cb.addItem("C++")
cb.addItems(["Java", "C#", "Python"])

print (cb.currentText())
print (cb.currentIndex())
```

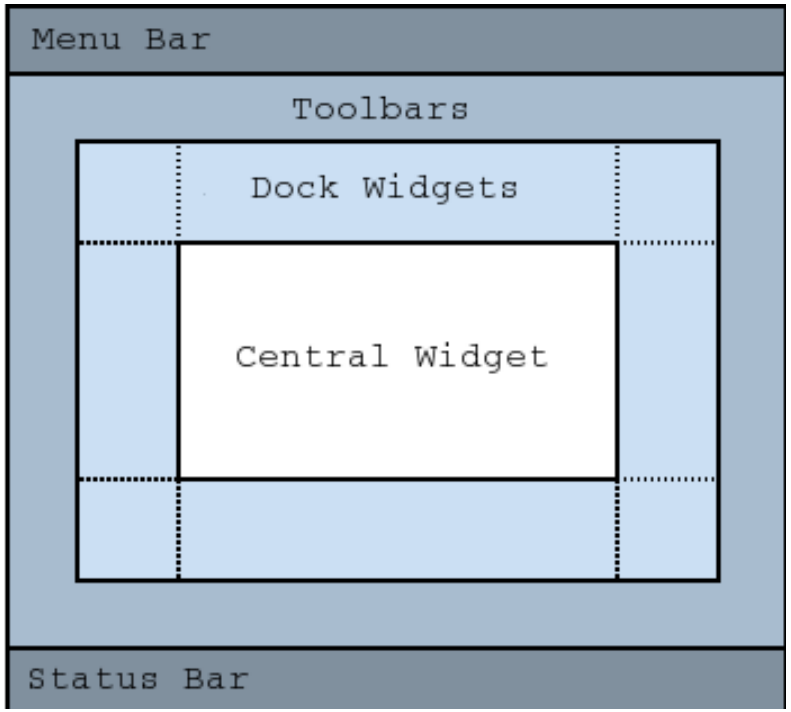
Signaux : *activated*, *currentIndexChanged*. Exemple :

```
cb.currentIndexChanged.connect(fonctSelectionChange)

def fonctSelectionChange():
    ....
```

Plan

Structure QMainWindow



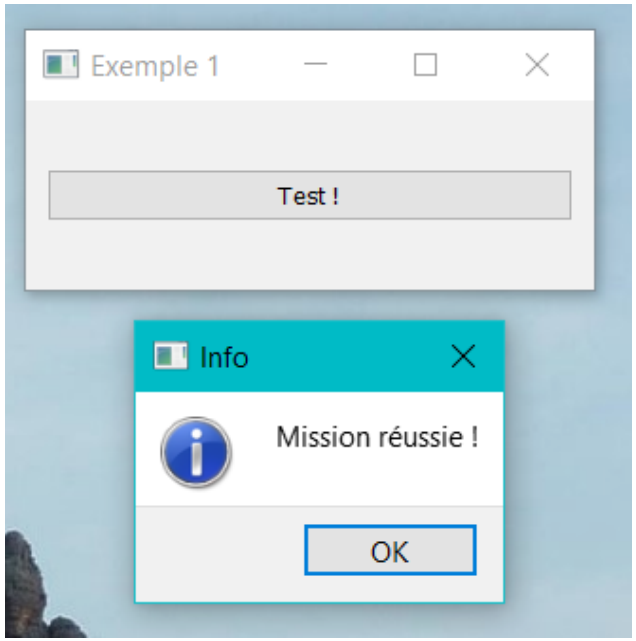
La plupart des applications Qt commencent au démarrage par une fenêtre principale et qui va donner accès à toutes les fonctionnalités de l'application. Une classe python qui hérite de *QMainWindow*

Structure QMainWindow

Les grandes étapes pour constituer une fenêtre principale en Qt :

1. créer une classe qui hérite de QMainWindow ;
2. [option sans *QtDesigner*]ajouter dans le constructeur les divers éléments graphiques constitutifs de la fenêtre : libellé, champs texte et boutons ;
3. définir dans la fenêtre principale les slots de réaction aux évènements et leurs associer ces méthodes ;
4. créer un programme principal qui instancie cette classe, l'application démarrera alors.

Exemple 1




```
import sys
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QMainWindow, QGridLayout
from PyQt5.QtWidgets import QWidget
from PyQt5.QtWidgets import QPushButton, QMessageBox
from PyQt5.QtCore import QSize
```

Exemple 1 sans Qt Designer

```
class BoutonWindow(QMainWindow):
    def __init__(self):
        QMainWindow.__init__(self)

        self.setMinimumSize(QSize(300, 100))
        self.setWindowTitle("Exemple 1")

        centralWidget = QWidget(self)
        self.setCentralWidget(centralWidget)

        gridLayout = QGridLayout(self)
        centralWidget.setLayout(gridLayout)

        boutonAfficher = QPushButton("Test !")
        gridLayout.addWidget(boutonAfficher, 0, 0)
        boutonAfficher.clicked.connect(self.affiche)

    def affiche(self):
        QMessageBox.information(self, "Info", "Mission réussie !
```

Exemple 1 sans Qt Designer

```
if __name__ == "__main__":
    def run_app():
        app = QtWidgets.QApplication(sys.argv)
        mainWin = BoutonWindow()
        mainWin.show()
        app.exec_()

    run_app()
```

Exemple 1 avec Qt Designer

```
from nomfichier import Ui_MainWindow

class TransformWindow(QMainWindow, Ui_MainWindow):
    def __init__(self, parent=None):
        super(TransformWindow, self).__init__(parent)
        self.setupUi(self)

        # juste la partie évènementielle

if __name__ == "__main__":
    def run_app():
        app = QtWidgets.QApplication(sys.argv)
        mainWin = BoutonWindow()
        mainWin.show()
        app.exec_()

    run_app()
```