Année universitaire 2016-2017
Université de Caen Basse-Normandie

Documentation PySide : Comment utiliser PySide pour notre IDE ?

Alexis Carreau
Thomas Lécluse
Emma Mauger
Théo Sarrazin
L2 Informatique

Table des matières

1 Introduction

Nous allons utiliser PySide pour faire notre interface graphique. PySide est une librairie qui fait l'intermédiaire entre Python et QT. QT est une librairie d'inferface graphique destinée au C. Nous avons fait ce choix car deux d'entre nous connaissaient déjà QT. De plus, il y a une documentation bien expliquée, explicite, et complète (cf https ://wiki.qt.io/PySide).

N'ayant pas fini notre projet, nous allons compléter cette documentation au fur et à mesure de l'avancement.

2 Fonctionnement

2.1 En théorie

Pour réaliser une interface graphique, on ré-implémente des classes de bases de QT, pour les adapter à nos besoins. Les classes créées héritent donc des superclasses et bénéficient de leurs attributs et de leurs méthodes. Cela nous permet de créer des widgets, qui sont des élements constituant l'interface et de les modifier à convenance.

2.2 En pratique

Quelques exemples :

Tout d'abord, il faut importer les modules nécessaires au bon fontionnemment de l'interface graphique :

```
import sys
from PySide.QtGui import *
```

FIGURE 1 – Importation des modules

Puis, on créé la classe Fenêtre qui hérite de la classe QWidget et on appelle le constructeur du parent.

```
class Fenetre(QWidget):
    def __init__(self, titre):
        super().__init__()
        self.setWindowTitle(titre)
```

FIGURE 2 – Définition de la structure de la classe Fenêtre

Ensuite, on fait appel à la classe QApplication pour instancier une nouvelle application. Puis, on affiche l'interface graphique avec notre classe Fenêtre qui hérite de la classe QWidget que nous avons créé.

```
app = QApplication(sys.argv)
fenetre = Fenetre("IDE de la mort qui tue (Bleu Royal)") #Creation of the main window
sys.exit(app.exec_())
```

FIGURE 3 – Création de l'interface

Un exemple d'utilisation de widget et le QTextEdit qui est une zone de texte éditable sur plusieurs lignes. On peut le personaliser en ajoutant par exmeple une police, la taille et la couleur du texte.

```
self.code = QTextEdit()  # Zone d'écriture du code
self.code.setFontFamily(self.police_code)  # Police d'écriture
self.code.setPalette(self.couleur_fond_code)  # Couleur de fond
self.code.setTextColor(self.couleur_ecriture_basique)  # Couleur d'écriture
self.code.setFontPointSize(self.taille_police)  # Taille de police
#self.code.setReadOnly(True)
```

FIGURE 4 – Création et personnalisation du QTextEdit

Un autre exemple d'utilisation de widget et le QPushButton qui est, comme son nom l'indique, un bouton qu'il faudra afficher plus tard et surtout connecter à une fonction pour qu'il est un rôle.

```
#self.img1 = QPixmap("Dragon.jpg")  # Image de lancement
self.ouvrir = QPushButton("Open")  # Bouton de lancement
#self.ouvrir.setIcon(QIcon(self.img1))  # Image sur le bouton
#self.ouvrir.setIconSize(QSize(self.code.width()*1.5, self.code.height()*1.5))  # Taille de l'image

# Bouton temporaire de sauvegarde
self.bouton_sauvegarde = QPushButton("Save")
```

FIGURE 5 – Création d'un bouton grâce au widget QPushButton

Une fois les objets créés, il faut les positionner. Pour cela, on utilise les Layout qui permet de positionner les différentes parties constituant notre interface graphique. Dans notre cas le self.layout est une instance de la classe QGridLayout. Enfin self.show permet d'afficher la fenêtre.

```
# Positionnement des Layouts
self.layout.addWidget(self.code, 0, 1, 6, 10)
self.layout.addWidget(self.ouvrir, 11, 0)
self.layout.addWidget(self.bouton_sauvegarde, 10, 0)

self.setLayout(self.layout)

self.show()
```

FIGURE 6 – Positionnement et affichage des widgets

Après avoir positionné et affiché les widgets, il est nécessaire, mais cela peut être fait avant, de connecter les widgets nécessaires à une fonction. C'est le cas du widget QPushButton introduit précédemment. En effet, pour connecter un bouton il faut utiliser la méthode connect() qui permet de faire le lien entre un clic dans une zone et une fonction. Dans l'image ci-dessous, la fonction bind prend en argument fenêtre et les boutons de sauvegarde et d'ouverture sont connectés à leur fonction respective.

```
def bind(fenetre):

    #binding
    fenetre.bouton_sauvegarde.clicked.connect(fenetre.save)
    fenetre.ouvrir.clicked.connect(fenetre.open)
```

FIGURE 7 – Connection d'un widget à sa fonction avec la méthode connect()