Développer des applications de bureau avec Python et Qt

Ahmed Ammar*

Feb 21, 2019

Table des matières

1 Introduction

Dans ce chapitre, nous allons passer à la création d'applications graphiques. De telles applications permettent de modifier l'apparence d'un programme en utilisant des **éléments de contrôle** tels que des **widgets**, des **boutons**, des **curseurs**, etc.

PyQt5 (`https://www.riverbankcomputing.com/software/pyqt/intro`) est une boîte à outils de widgets d' interface graphique (en anglais **GUI** pour *Graphical User Interface*) combinant le langage de programmation Python et le logiciel **Qt5** (`https://www.qt.io`). PyQt5 est une bibliothèque d'interface graphique populaire qui présente de nombreux avantages par rapport aux autres bibliothèques d'interface graphique telles que *Tkinter* et *wx*. Ceux-ci inclus :

*Email : `ahmed.ammar@fst.utm.tn`. Faculté des Sciences de Tunis, Université de Tunis El Manar.

— Bibliothèque d'interface graphique multi-plateforme (Windows, MacOS, Linux).
— Bonne performance.
— Prise en charge des styles personnalisés.
— Plus de bibliothèques pour la conception d'interface graphique complexe.
— Facilité d'utilisation.

2 Premières étapes pour la création d'une application graphique à l'aide de PyQt5

2.1 Importation de PyQt5 et création d'une "fenêtre PyQt5"

Passons directement à l'action et apprenons à créer une fenêtre simple avec PyQt5. Premièrement, nous devrons importer certains modules essentiels à l'exécution d'une interface graphique avec PyQt5. Nous commençons par **importer** quelques sous-modules de PyQt5.

```
from PyQt5.QtWidgets import QApplication, QWidget
```

Notice
`QtWidgets` est l'un des nombreux composants de PyQt5. Certains des plus couramment utilisés sont énumérés ci-dessous :
— `QtWidgets` : Contient des classes qui fournissent un ensemble d'éléments pour créer une interface graphique classique de type bureau.
— `QtCore` : Ce module contient les classes principales, y compris la boucle d'événement et les mécanismes `signal` et `slot` de Qt.
— `QtGui` : Celui-ci contient des classes pour l'intégration du système de fenêtrage, la gestion des événements, les graphiques 2D, les images de base, les polices et le texte.
— `QtDesigner` : Ce module contient des classes permettant l'extension de *Qt Designer* à l'aide de PyQt5.
— `uic` : Ce module contient des classes permettant de gérer les fichiers **.ui** créés par Qt Designer décrivant l'ensemble ou une partie d'une interface graphique.
— etc.

Ensuite, nous importons le module `sys` car nous voulons accéder aux arguments de ligne de commande. Ceux-ci sont contenus dans la liste 'sys.argv

```
import sys
app = QApplication(sys.argv)
```

En utilisant les arguments de ligne de commande contenus dans `sys.argv`, nous devrions créer un objet `QApplication`. Dans l'exemple ci-dessus, nous

avons enregistré cet objet sous le nom **app**. Ceci est l'objet *exécuté* lorsque nous exécutons à l'invite de commande la commande `python filename.py` où **filename.py** est le fichier dans lequel les instructions sont stockées. Cet objet contiendra tous les éléments de l'interface graphique et leurs méthodes. Par conséquent, lorsque cet objet est créé et *exécuté*, nous avons accès à toutes les interfaces graphiques utilisées dans le programme. Les éléments de l'interface graphique sont hérités de la classe `QWidgets`. Nous obtenons donc un objet `QWidget` comme indiqué ci-dessous.

```
window = QWidget()
```

En utilisant les nombreuses méthodes *set*, nous pouvons définir les valeurs des différents attributs de cet objet *window*. Nous allons d'abord définir les dimensions de la fenêtre en utilisant la méthode `setGeometry`. Il prend quatre paramètres : **x** coordonnée du point le plus à gauche de l'objet (*window*), **y** coordonnée du point le plus haut, **la largeur** et **la hauteur** de la fenêtre, dans cet ordre. Toutes ces valeurs doivent être des entiers référencés par rapport aux coordonnées d'écran natives. Par exemple, nous pourrions écrire :

```
window.setGeometry(400, 100, 300, 200)
```

Cela définirait la fenêtre avec une largeur de 300 pixels et une hauteur de 200 pixels à 400 pixels du côté gauche et à 100 pixels du haut de l'écran natif (écran de votre ordinateur).

Nous pouvons définir un titre pour la fenêtre. Par exemple :

```
window.setWindowTitle('My first app')
```

afficherait *My first app* dans la barre de titre de la fenêtre.

Une fois que nous sommes satisfaits de l'interface graphique que nous avons construite, nous appelons :

```
window.show()
```

pour afficher l'objet graphique dans *l'application* que nous avons créée. Cependant, l'application n'est toujours pas exécutée. Pour exécuter l'application, nous exécutons la commande :

```
app.exec()
```

La méthode `.show ()` est une méthode QT qui ouvre la fenêtre à l'écran pour l'utilisateur.

Enfin, pour assurer une fermeture agréable et propre de l'application lorsque nous quittons :

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

Le code complet de cette application est donné ci-dessous et le résultat est illustré à la Figure 1. Le code peut être stocké dans un fichier, par exemple **myfirstapp.py**, puis lorsque vous exécutez la commande `python MyFirstApp.py` à l'invite de commande (ou dans Spyder), vous obtenez la fenêtre de sortie affichée.

```
# NOM DU FICHIER: myfirstapp.py
#% IMPORTATION
from PyQt5.QtWidgets import QApplication, QWidget
import sys
# créer un objet QApplication
app = QApplication(sys.argv)
# appeler la classe QWidgets
window = QWidget()
# définir les dimensions de la fenêtre
window.setGeometry(400, 100, 300, 200)
# définir un titre pour la fenêtre
window.setWindowTitle('My first app')
# afficher l'objet graphique dans l'application
window.show()
# fermer l'application
sys.exit(app.exec_())
```

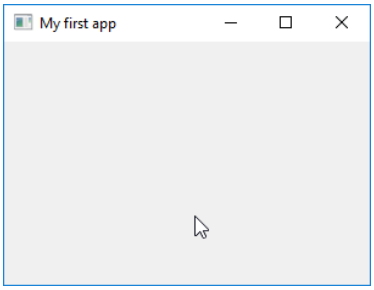

FIGURE 1 – Une fenêtre PyQt5 créée par le code `myfirstapp.py` et également par le code `myfirstappStructure.py`.

2.2 Structure d'application PyQT

Les étapes suivantes (dont certaines ont déjà été utilisées dans l'exemple ci-dessus) aideront à créer une structure solide et propre pour toute application développée avec PyQt :

1. Importer les modules nécessaires.
2. Créer une classe héritée de `QMainWindow` (à l'aide de la construction `class MainWindow(QMainWindow):`). La `QMainWindow` est une classe Qt qui fournit une fenêtre principale de l'application avec toutes les options (telle que la barre de menus, la barre d'état, etc.).

3. Implémentez le **constructeur** (également appelé **l'initialiseur**) de la classe en utilisant la méthode `__init __ ()`. Chaque fois qu'un objet d'une classe est créé, cette méthode sera exécutée. L'initialiseur commence généralement par un appel à l'initialiseur de la classe `super ()` ; Ainsi, toutes les méthodes définies dans la classe parent sont immédiatement disponibles, même si elles sont remplacées dans la définition de classe actuelle.
4. Cette opération est ensuite suivie par la définition des valeurs (par défaut) pour divers attributs, tels que la géométrie, le titre, etc., et la définition des différentes méthodes.
5. Enfin, utiliser la construction `if __name__ == "__main __":` à la fin pour créer un objet `QApplication` et l'exécuter.

La fenêtre illustrée à la Figure 1 peut également être créée à l'aide des étapes décrites ci-dessus. Le code permettant de faire cela est donné dans le script **myfirstappStructure.py**. Dans une fenêtre d'invite de commande, exécutez la commande `python myfirstappStructure.py` (ou exécutez le script sur Spyder), où **myfirstappStructure.py** est le nom du fichier contenant le code permettant d'afficher la fenêtre illustrée à la Figure 1.

```python
# NOM DU FICHIER: myfirstappStructure.py
#% IMPORTATION
from PyQt5.QtWidgets import QApplication, QMainWindow

class MainWindow(QMainWindow):
    '''
    DOCUMENTATION
    -------------
    Créer une fenêtre (300x200 pixels)
    '''
    def __init__(self):
        '''
        INITIALISEUR
        ------------
        La classe `MainWindow()` rendre une fenêtre (300x200 pixels)
        avec un titre 'My first app'.
        '''
        super(MainWindow, self).__init__()
        # Propriétés de l'interface graphique
        self.setGeometry(400, 100, 300, 200)
        self.setWindowTitle('My first app')

if __name__ == "__main__":
    import sys
    app = QApplication(sys.argv)
    MyApplication = MainWindow()
    MyApplication.show()
    sys.exit(app.exec_())
```

2.3 Widgets, événements et signaux

Les widgets sont les éléments de base d'une interface graphique. PyQt5 a une large gamme de widgets, y compris des boutons (`buttons`), des étiquettes

(`labels`), des cases à cocher (`check boxes`), des curseurs (`sliders`), des zones de liste (`list boxes`), etc. Nous décrirons plusieurs widgets couramment utilisés dans cette section. Ceux-ci incluent un `QPushButton`, un `ToggleButton`, un `QLabel`, un `QCheckBox`, un `QSlider`, un `QProgressBar`, un `QCalendarWidget`, etc.

Ajouter des étiquettes : `QLabel`. Nous avons besoin d'une étiquette pour afficher les messages. Un `QLabel` peut afficher du texte brut ou riche, HTML. Le code présenté ci-dessous (**myfirstappLabels.py**) ajouterait deux objets `QLabel` à la même fenêtre que celle précédemment créée. Le premier affiche un texte brut (`"Ma première application affiche: "`) et le second un texte HTML enrichi ("Bonjour IE3!", Gras et de couleur verte). L'exécution du code produira une fenêtre illustrée à la Figure 2.

```
# NOM DU FICHIER: myfirstappLabels.py
#% IMPORTATION
from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel

class MainWindow(QMainWindow):
    '''
    DOCUMENTATION
    -------------
    Créer une fenêtre (300x200 pixels)
    '''
    def __init__(self):
        '''
        INITIALISEUR
        ------------
        La classe `MainWindow()` rendre une fenêtre (300x200 pixels)
        avec un titre 'My first app'.
        '''
        super(MainWindow, self).__init__()
        # GUI proprieties
        self.setGeometry(400, 100, 300, 200)
        self.setWindowTitle('My first app')
        # Ajouter des étiquettes: QLabel
        ## Label 1: Texte brut
        label1 = QLabel('Ma première application affiche: ', self)
        # fixer la largeur de l'étiquette
        label1.setFixedWidth(160)
        # position de l'étiquette (x, y) en pixels
        label1.move(10, 10)
        ## Label 2: Texte riche
        label2 = QLabel("", self)
        # créer un message HTML
        message = "<h3><b><font color='green'>Bonjour IE3!</font></b>"
        # ajouter le text du message
        label2.setText(message)
        # fixer la largeur de l'étiquette
        label2.setFixedWidth(120)
         # position de l'étiquette (x, y) en pixels
        label2.move(200, 10)

if __name__ == "__main__":
    import sys
    app = QApplication(sys.argv)
```

```
    MyApplication = MainWindow()
    MyApplication.show()
    sys.exit(app.exec_())
```

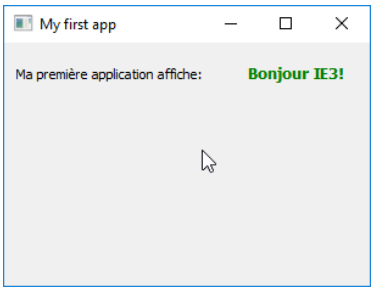

FIGURE 2 – Résultat de l'exécution du code `myfirstappLabels.py`.

Ajouter des boutons : `QPushButton`. La plupart des interfaces graphiques sont utilisées pour prendre les entrées de l'utilisateur et produire des sorties en fonction de l'entrée donnée. Une technique courante consiste à demander à l'utilisateur de cliquer sur un bouton `OK` ou sur un bouton `Annuler`. Dans PyQt5, cela est implémenté à l'aide de la classe `QPushButton`. Chaque widget d'une application peut être programmé pour répondre de manière prédéterminée à tout changement survenant dans la fenêtre de l'application ou n'importe où sur l'ordinateur. Ces *changements* sont appelés **des événements**. Par exemple, nous pourrions déplacer la souris autour de l'écran ou sur la fenêtre de l'application, ou peut-être avons-nous appuyé sur la touche `Entrée`. Lorsqu'un tel événement se produit, le système émet un ou plusieurs signaux ; autrement dit, modifie certaines propriétés des widgets. Nous pouvons écrire du code qui sera exécuté à chaque fois qu'un tel signal est émis. Ce morceau de code s'appelle un **slot**. Par exemple, nous pouvons souhaiter fermer la fenêtre et quitter l'application lorsque le bouton gauche d'une souris est cliqué sur le bouton de commande `Annuler`. Lorsque cet événement de clic gauche de la souris sur le bouton `Annuler` se produit, un signal de **clicked** est émis sur le bouton et un **slot** qui quitte l'application et quitte sera implémenté. Cependant, pour que cela se produise, nous devons explicitement **connecter** le **slot** au signal. Toutes ces idées sont montrées dans le code **myfirstappButtons.py**.

```
# NOM DU FICHIER: myfirstappButtons.py
#% IMPORTATION
from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel, QPushButton
from PyQt5.QtCore import pyqtSlot


class MainWindow(QMainWindow):
    '''
    DOCUMENTATION
    -------------
```

```
    Créer une fenêtre (300x200 pixels)
    '''
    def __init__(self):
        '''
        INITIALISEUR
        ------------
        La classe `MainWindow()` rendre une fenêtre (300x200 pixels)
        avec un titre 'My first app'.
        '''
        super(MainWindow, self).__init__()
        # GUI proprieties
        self.setGeometry(400, 100, 300, 200)
        self.setWindowTitle('My first app')
        # Ajouter des étiquettes: QLabel
        ## Label 1: Texte brut
        self.label1 = QLabel('Ma première application affiche: ', self)
        # fixer la largeur de l'étiquette
        self.label1.setFixedWidth(160)
        # position de l'étiquette (x, y) en pixels
        self.label1.move(10, 10)
        ## Label 2: Texte riche
        self.label2 = QLabel("", self)
        # créer un message HTML
        message = "<h3><b><font color='green'>Bonjour IE3!</font></b>"
        # ajouter le text du message
        self.label2.setText(message)
        # fixer la largeur de l'étiquette
        self.label2.setFixedWidth(120)
         # position de l'étiquette (x, y) en pixels
        self.label2.move(200, 10)

        # Ajouter des boutons: `QPushButton`
        # Push Button 1: Bouton bonjour
        button1 = QPushButton('Bonjour', self)
        button1.setToolTip("C'est le bouton Bonjour")
        button1.move(50, 50)
        # connecter le signal à l'événement
        button1.clicked.connect(self.on_click_button1)
        # Push Button 2: Bouton au revoir
        button2 = QPushButton('Au revoir', self)
        button2.setToolTip("C'est le bouton Au revoir")
        button2.move(170, 50)
        # connecter le signal à l'événement
        button2.clicked.connect(self.on_click_button2)

    @pyqtSlot()  # signal du bouton bonjour
    def on_click_button1(self):
        '''
        Afficher un message "Bonjour"
        '''
        message = "<h3><b><font color='green'>Bonjour IE3!</font></b>"
        self.label2.setText(message)
        self.label2.setFixedWidth(120)

    @pyqtSlot()  # signal du bouton au revoir
    def on_click_button2(self):
        '''
        Afficher un message "Au revoir"
        '''
        message = "<h3><b><font color='red'>Au revoir IE3!</font></b>"
```

```
        self.label2.setText(message)
        self.label2.setFixedWidth(120)

if __name__ == "__main__":
    import sys
    app = QApplication(sys.argv)
    MyApplication = MainWindow()
    MyApplication.show()
    sys.exit(app.exec_())
```

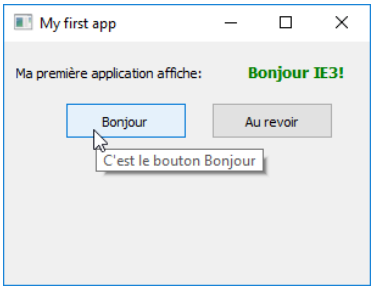

FIGURE 3 – Résultat de l'exécution du code `myfirstappButtons.py`.

Toute méthode peut être connectée à un signal. Cependant, pour être sûr, il est préférable de s'assurer qu'une méthode particulière est bien un **slot**. Ceci est réalisé en déclarant une méthode comme étant un slot en utilisant le décorateur `@pyqtSlot()` comme indiqué dans le code **myfirstappButtons.py**.

Ajouter des Spin Box : `QSpinBox`. Les Spin Box sont des widgets utilisés pour accepter et afficher des nombres entiers. Celles-ci peuvent être considérées comme une boîte d'édition avec une molette attachée. En cliquant une fois sur une flèche vers le haut de la molette, la valeur dans la zone d'édition augmente d'un point ; cliquer sur la flèche vers le bas diminuera la valeur de un. Vous pouvez également entrer la valeur directement dans la zone d'édition. La plage de valeurs accessible dans un spinbox peut être définie à l'aide de la méthode `setRange()` ou de la combinaison des méthodes `setMinimum()` et `setMaximum()`. Toutes ces idées sont montrées dans le code **myfirstappSpinBox.py** et illustrées par la Figure 4.

```
# NOM DU FICHIER: myfirstappSpinBox.py
#% IMPORTATION
from PyQt5.QtWidgets import (QApplication, QMainWindow, QLabel,
                             QPushButton, QSpinBox)
from PyQt5.QtCore import pyqtSlot


class MainWindow(QMainWindow):
    '''
    DOCUMENTATION
    -------------
    Créer une fenêtre (300x200 pixels)
```

```
    '''
    def __init__(self):
        '''
        INITIALISEUR
        ------------
        La classe `MainWindow()` rendre une fenêtre (300x200 pixels)
        avec un titre 'My first app'.
        '''
        super(MainWindow, self).__init__()
        # GUI proprieties
        self.setGeometry(400, 100, 300, 200)
        self.setWindowTitle('My first app')
        # Ajouter des étiquettes: QLabel
        ## Label 1: Texte brut
        self.label1 = QLabel('Ma première application affiche: ', self)
        # fixer la largeur de l'étiquette
        self.label1.setFixedWidth(160)
        # position de l'étiquette (x, y) en pixels
        self.label1.move(10, 10)
        ## Label 2: Texte riche
        self.label2 = QLabel("", self)
        # créer un message HTML
        message = "<h3><b><font color='green'>Bonjour IE3!</font></b>"
        # ajouter le text du message
        self.label2.setText(message)
        # fixer la largeur de l'étiquette
        self.label2.setFixedWidth(120)
         # position de l'étiquette (x, y) en pixels
        self.label2.move(200, 10)

        # Ajouter des boutons: `QPushButton`
        # Push Button 1: Bouton bonjour
        button1 = QPushButton('Bonjour', self)
        button1.setToolTip("C'est le bouton Bonjour")
        button1.move(50, 50)
        # connecter le signal à l'événement
        button1.clicked.connect(self.on_click_button1)
        # Push Button 2: Bouton au revoir
        button2 = QPushButton('Au revoir', self)
        button2.setToolTip("C'est le bouton Au revoir")
        button2.move(170, 50)
        # connecter le signal à l'événement
        button2.clicked.connect(self.on_click_button2)

        # Ajouter des Spin Box: QSpinBox
        self.spb = QSpinBox(self)
        # Spin Box changer une valeur entière de 0 à 100
        self.spb.setMinimum(0)
        self.spb.setMaximum(100)
        self.spb.setValue(50) # valeur par défaut
        self.spb.setSingleStep(1) # pas
        # définir la géométrie (x, y, largeur, hauteur) en pixels
        self.spb.setGeometry(120, 100, 50, 20)
        # connecter le signal à l'événement
        self.spb.valueChanged.connect(self.spb_valuechange)
        # définir une étiquette disant à quoi Spin Box fait référence
        txt = QLabel('La valeur actuelle est: ', self)
        # définir la géométrie (x, y, largeur, hauteur) en pixels
        txt.setGeometry(10, 100, 100, 20)
        # définir une étiquette pour recevoir une valeur du spin box
        self.val = QLabel("", self)
```

```
        # définir la géométrie (x, y, largeur, hauteur) en pixels
        self.val.setGeometry(190, 100, 30, 20)

    @pyqtSlot()  # signal du bouton bonjour
    def on_click_button1(self):
        '''
        Afficher un message "Bonjour"
        '''
        message = "<h3><b><font color='green'>Bonjour IE3!</font></b>"
        self.label2.setText(message)
        self.label2.setFixedWidth(120)

    @pyqtSlot()  # signal du bouton au revoir
    def on_click_button2(self):
        '''
        Afficher un message "Au revoir"
        '''
        message = "<h3><b><font color='red'>Au revoir IE3!</font></b>"
        self.label2.setText(message)
        self.label2.setFixedWidth(120)

    @pyqtSlot(int)  # signal du Spin Box: renvoie un entier
    def spb_valuechange(self, value):
        '''
        la valeur du Spin Box est un entier,
        convertissez-le en chaîne à l'aide de la fonction str().
        '''
        self.val.setText(str(self.spb.value()))

if __name__ == "__main__":
    import sys
    app = QApplication(sys.argv)
    MyApplication = MainWindow()
    MyApplication.show()
    sys.exit(app.exec_())
```

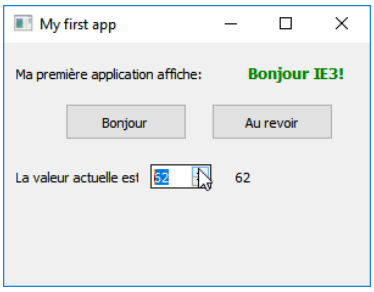

FIGURE 4 – Résultat de l'exécution du code `myfirstappSpinBox.py`.

Ajouter des curseurs : `QSlider`. Un QSlider est un widget doté d'une poignée simple. Cette poignée peut être déplacée d'avant en arrière. En utilisant cela, nous pouvons choisir une valeur pour une tâche spécifique. Comme le Spin Box, un curseur ne peut avoir qu'une valeur entière ; la plage des valeurs peut

être définie entre un minimum et un maximum. Parfois, l'utilisation d'un curseur est plus naturelle que celle d'un Spin Box et cela dépend de la tâche ou de l'application.

```
# NOM DU FICHIER: myfirstappSlider.py
#% IMPORTATION
from PyQt5.QtWidgets import (QApplication, QMainWindow, QLabel,
                             QPushButton, QSpinBox, QSlider)
from PyQt5.QtCore import pyqtSlot, Qt


class MainWindow(QMainWindow):
    '''
    DOCUMENTATION
    -------------
    Créer une fenêtre (300x200 pixels)
    '''
    def __init__(self):
        '''
        INITIALISEUR
        ------------
        La classe `MainWindow()` rendre une fenêtre (300x200 pixels)
        avec un titre 'My first app'.
        '''
        super(MainWindow, self).__init__()
        # GUI proprieties
        self.setGeometry(400, 100, 300, 200)
        self.setWindowTitle('My first app')
        # Ajouter des étiquettes: QLabel
        ## Label 1: Texte brut
        self.label1 = QLabel('Ma première application affiche: ', self)
        # fixer la largeur de l'étiquette
        self.label1.setFixedWidth(190)
        # position de l'étiquette (x, y) en pixels
        self.label1.move(10, 10)
        ## Label 2: Texte riche
        self.label2 = QLabel("", self)
        # créer un message HTML
        message = "<h3><b><font color='green'>Bonjour IE3!</font></b>"
        # ajouter le text du message
        self.label2.setText(message)
        # fixer la largeur de l'étiquette
        self.label2.setFixedWidth(120)
         # position de l'étiquette (x, y) en pixels
        self.label2.move(200, 10)

        # Ajouter des boutons: `QPushButton`
        # Push Button 1: Bouton bonjour
        button1 = QPushButton('Bonjour', self)
        button1.setToolTip("C'est le bouton Bonjour")
        button1.move(50, 50)
        # connecter le signal à l'événement
        button1.clicked.connect(self.on_click_button1)
        # Push Button 2: Bouton au revoir
        button2 = QPushButton('Au revoir', self)
        button2.setToolTip("C'est le bouton Au revoir")
        button2.move(170, 50)
        # connecter le signal à l'événement
        button2.clicked.connect(self.on_click_button2)
```

```
        # Ajouter des Spin Box: QSpinBox
        self.spb = QSpinBox(self)
        # Spin Box changer une valeur entière de 0 à 100
        self.spb.setMinimum(0)
        self.spb.setMaximum(100)
        self.spb.setValue(50) # valeur par défaut
        self.spb.setSingleStep(1) # pas
        # définir la géométrie (x, y, largeur, hauteur) en pixels
        self.spb.setGeometry(120, 100, 50, 20)
        # connecter le signal à l'événement
        self.spb.valueChanged.connect(self.spb_valuechange)
        # définir une étiquette disant à quoi Spin Box fait référence
        txt = QLabel('La valeur actuelle est: ', self)
        # définir la géométrie (x, y, largeur, hauteur) en pixels
        txt.setGeometry(10, 100, 100, 20)
        # définir une étiquette pour recevoir une valeur du spin box
        self.val = QLabel("", self)
        # définir la géométrie (x, y, largeur, hauteur) en pixels
        self.val.setGeometry(190, 100, 30, 20)

        # Slider: Slider change integer value from 0 to 100
        self.sldr = QSlider(Qt.Horizontal, self)
        self.sldr.setMinimum(0)
        self.sldr.setMaximum(100)
        self.sldr.setValue(50)
        self.sldr.setSingleStep(1)
        # set geometry (x,y,width,hight)
        self.sldr.setGeometry(10, 150, 280, 20)
        self.sldr.valueChanged.connect(self.sldr_valuechange)

    @pyqtSlot()  # signal du bouton bonjour
    def on_click_button1(self):
        '''
        Afficher un message "Bonjour"
        '''
        message = "<h3><b><font color='green'>Bonjour IE3!</font></b>"
        self.label2.setText(message)
        self.label2.setFixedWidth(120)

    @pyqtSlot()  # signal du bouton au revoir
    def on_click_button2(self):
        '''
        Afficher un message "Au revoir"
        '''
        message = "<h3><b><font color='red'>Au revoir IE3!</font></b>"
        self.label2.setText(message)
        self.label2.setFixedWidth(120)

    @pyqtSlot(int)  # signal du Spin Box: renvoie un entier
    def spb_valuechange(self, value):
        '''
        la valeur du Spin Box est un entier,
        convertissez-le en chaîne à l'aide de la fonction str().
        '''
        self.val.setText(str(self.spb.value()))
        # connecter le curseur au Spin Box
        self.sldr.setValue(value)

    @pyqtSlot(int)  # signal du curseur
    def sldr_valuechange(self, value):
        '''
```

```
        connecter le Spin Box au curseur
        '''
        self.spb.setValue(value)


if __name__ == "__main__":
    import sys
    app = QApplication(sys.argv)
    MyApplication = MainWindow()
    MyApplication.show()
    sys.exit(app.exec_())
```

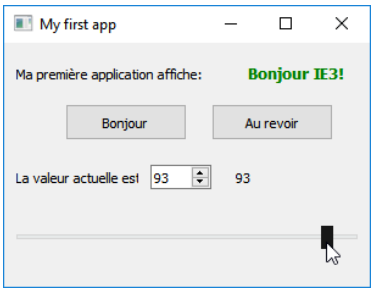

FIGURE 5 – Résultat de l'exécution du code `myfirstappSlider.py`.

3 Le concepteur Qt : "Qt Designer"

Le paquet PyQt est livré avec un outil de générateur d'interface graphique appelé Qt Designer (ou concepteur Qt). Qt Designer est l'outil Qt permettant de concevoir et de construire des interfaces graphiques à l'aide d'une simple approche **glisser-déposer**. Il vous permet de concevoir des widgets, des boîtes de dialogue ou des fenêtres principales complètes à l'aide de formulaires à l'écran (en anglais appelés : on-screen forms). Il a la possibilité de prévisualiser les conceptions pour s'assurer que l'interface graphique fonctionne comme prévu. Il permet une interface de prototypage avant qu'un code ne soit écrit.

Qt Designer peut être démarré en tapant `designer` dans la fenêtre d'invite de commande sous Windows ou sur un terminal sous n'importe quel système Linux. Sur un Mac, double-cliquez sur l'application Qt Desginer dans le répertoire "anaconda3 / bin" pour lancer Qt Designer. Une fois démarré, Qt Designer ressemblera à celui illustré à la figure 6.

La création d'une interface graphique à l'aide de Qt Designer commence par la sélection d'une fenêtre de niveau supérieur pour l'application. Dans un premier temps, Qt Designer affiche une boîte de dialogue d'initialisation dans laquelle vous pouvez choisir le type de base de l'interface graphique que vous souhaitez créer. Vous avez la possibilité de choisir entre trois types de formulaires : **Dialog**, **Main Window** et **Widget**.

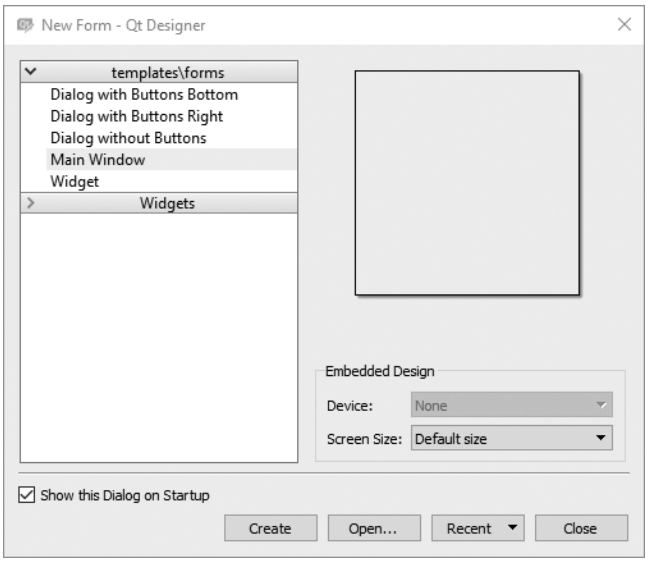

FIGURE 6 – Dialogue d'initialisation du concepteur Qt.

Une fois que le type de formulaire requis est sélectionné et créé (en cliquant sur le bouton **Create**), il peut être redimensionné pour répondre aux besoins de l'application et tous les widgets peuvent être placés dessus.

3.1 La fenêtre du concepteur Qt

Par défaut, à gauche du formulaire se trouve un panneau appelé la *Boîte à widgets* (Widget Box) qui contient tous les objets Qt regroupés sous différentes catégories. Sur la droite, de nombreux panneaux tels que *l'inspecteur d'objets* (Object Inspector), *l'éditeur de propriétés* (Property Editor), etc.

3.2 L'éditeur de propriété

À suivre...

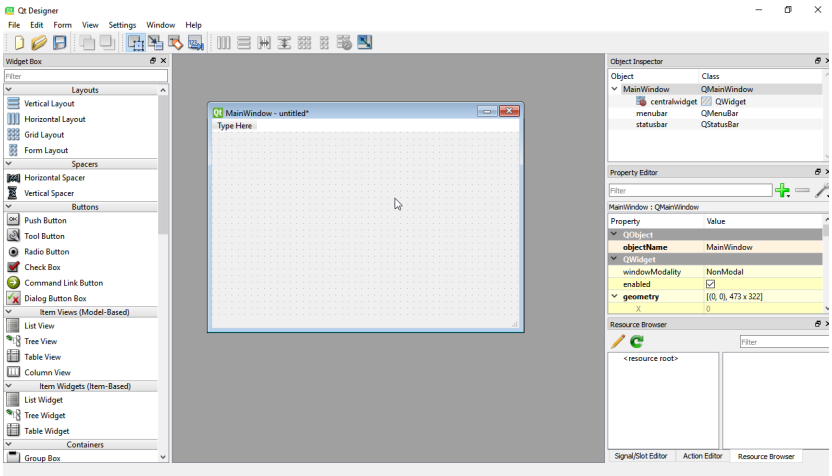

FIGURE 7 – La fenêtre du concepteur Qt.

FIGURE 8 – Interface graphique créée par le concepteur Qt.