eBook Gratuit

APPRENEZ pyqt5

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#pyqt5

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: pyqt5

It is an unofficial and free pyqt5 ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official pyqt5.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec pyqt5

Remarques

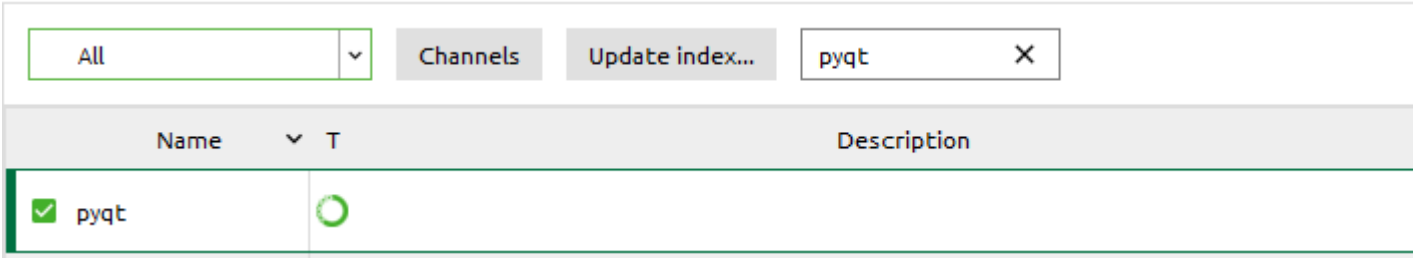
Cette section fournit une vue d'ensemble de ce qu'est pyqt5 et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans pyqt5, et établir un lien avec les sujets connexes. La documentation de pyqt5 étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

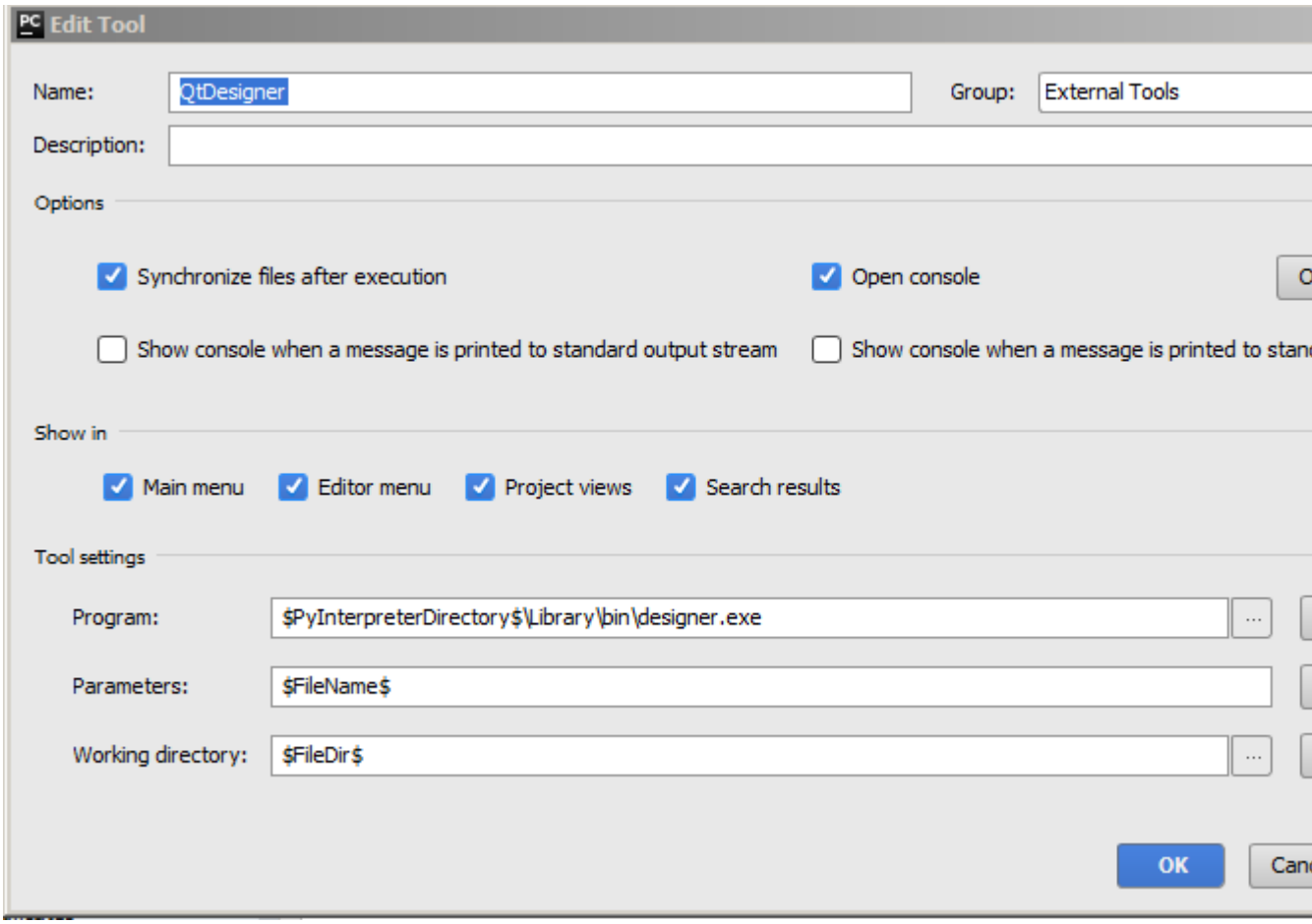
Examples

Installation ou configuration

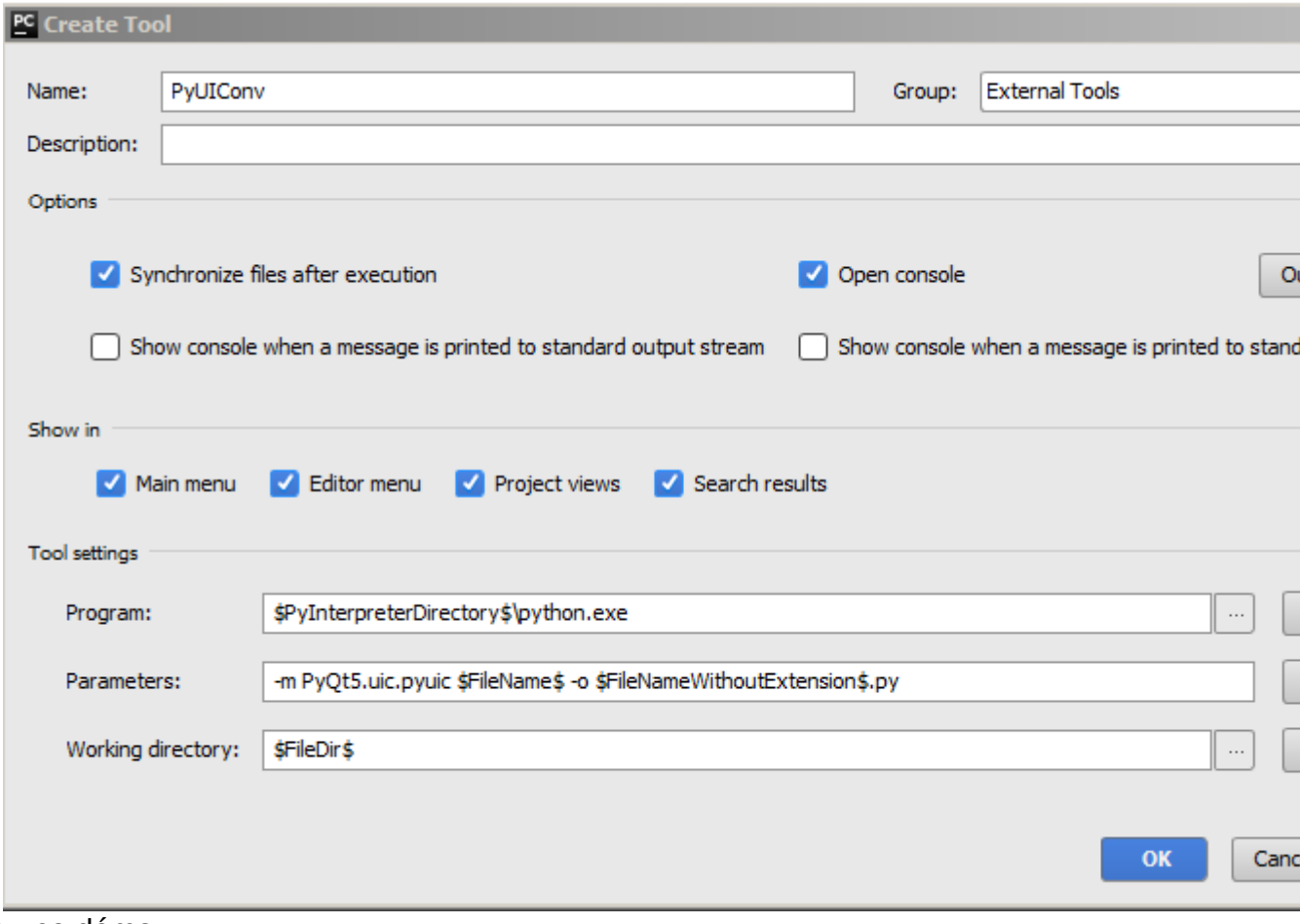
1. Installez Anaconda (PyQt5 est intégré), en particulier pour les utilisateurs de Windows.



2. Intégrer QtDesigner et QtUIConvert dans PyCharm (outils externes)
    - Ouvrir les `Settings` PyCharm> `Tools` > `External Tools`
    - Create Tool (QtDesigner) - utilisé pour éditer les fichiers * .ui

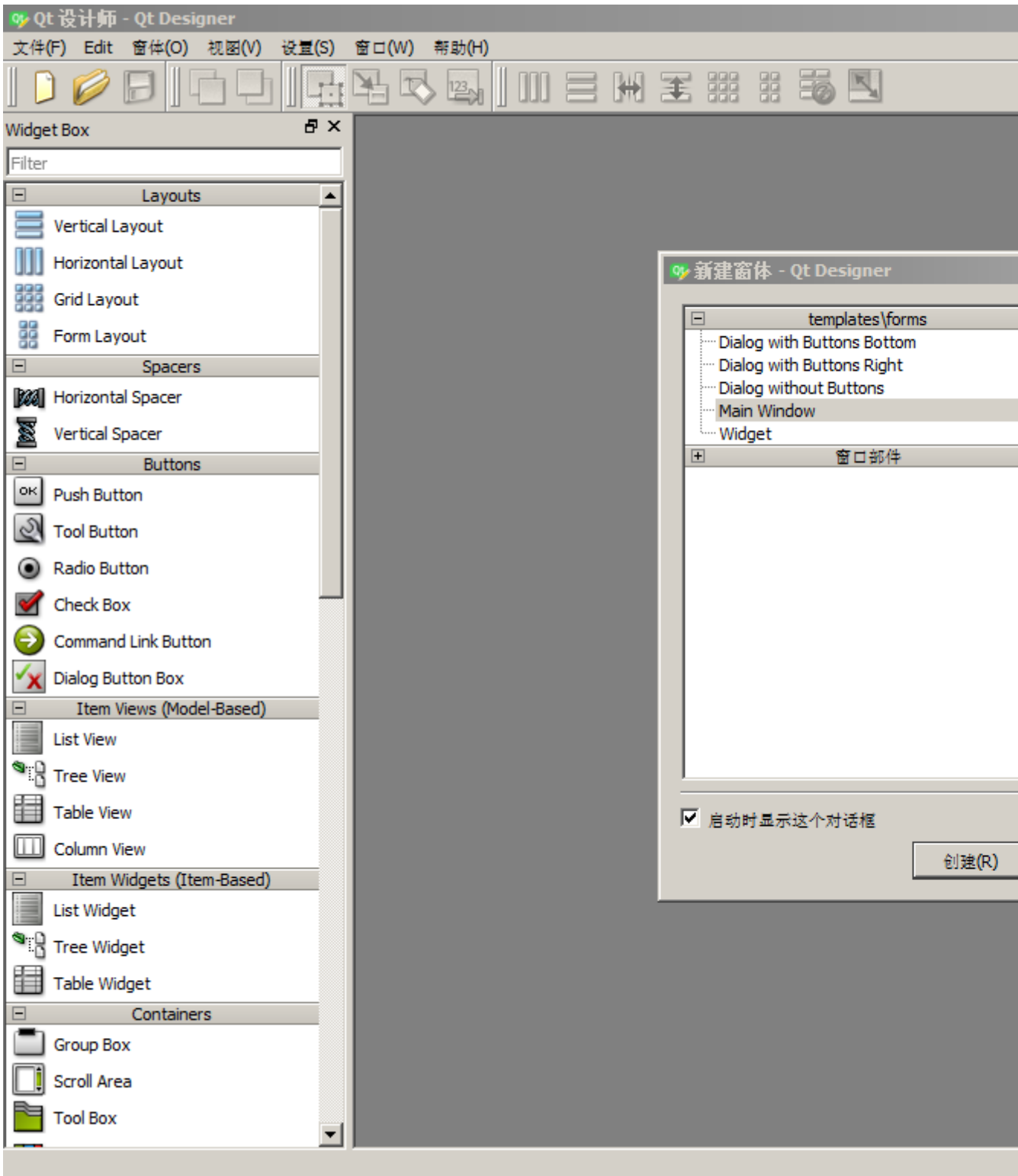


- Create Tool (PyUIConv) - utilisé pour convertir * .ui en * .py

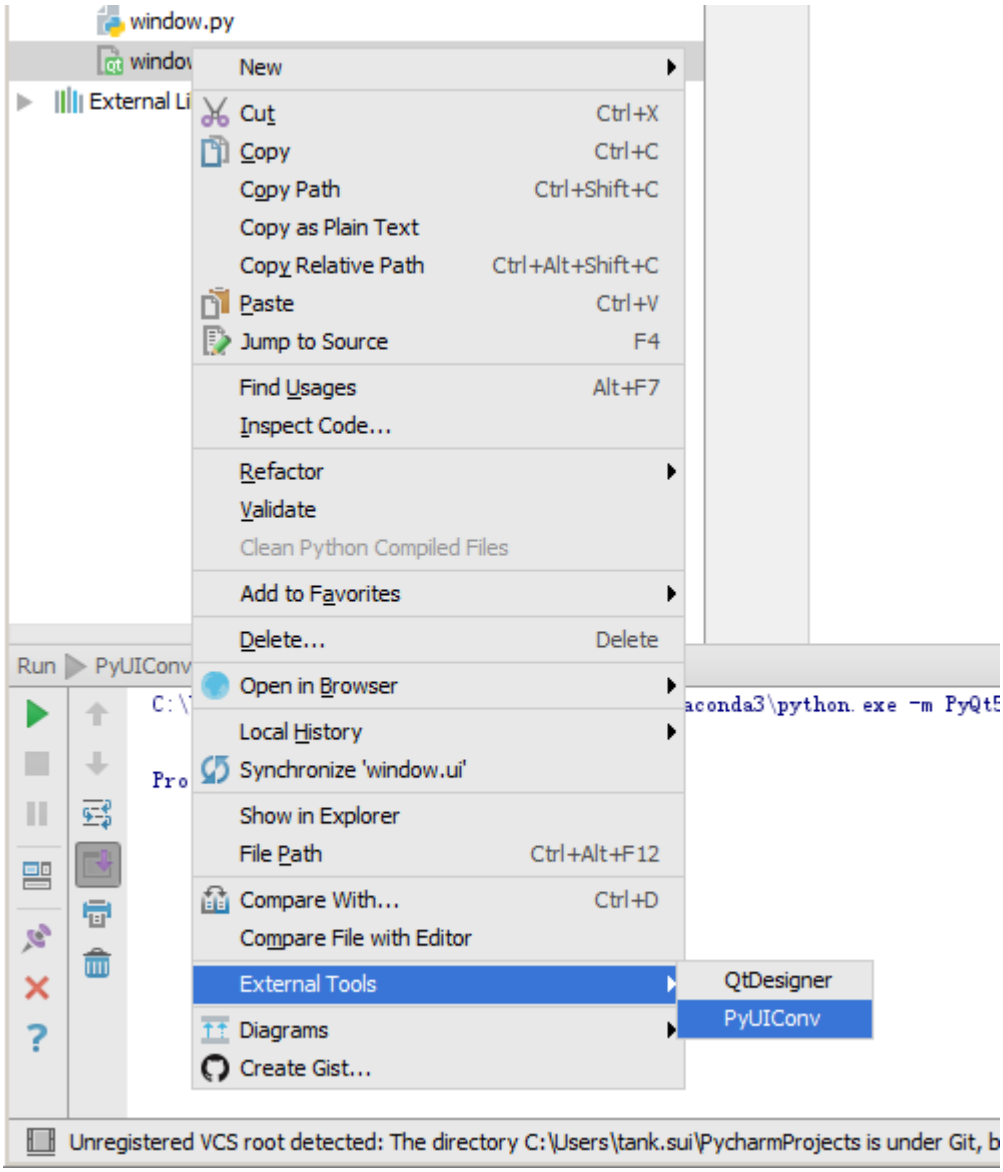


3. Ecrire une démo

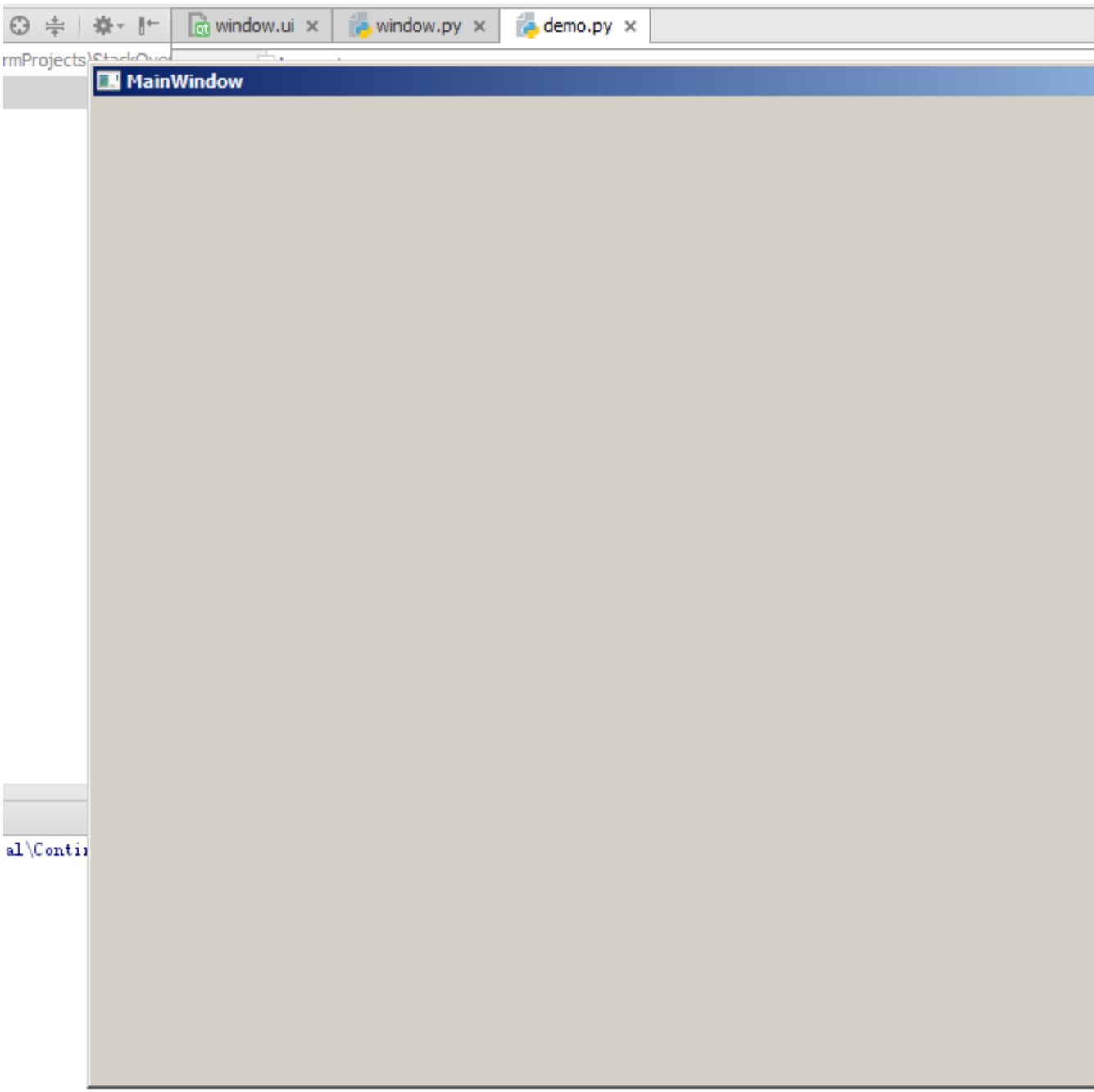
- new window.ui par un outil externe (QtDesigner)



- convertir en window.py par un outil externe (PyUIConv)


window.py
External Li
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy as Plain Text
Copy Relative Path Ctrl+Alt+Shift+C
Paste Ctrl+V
Jump to Source F4
Find Usages Alt+F7
Inspect Code...
Refactor
Validate
Clean Python Compiled Files
Add to Favorites
Delete... Delete
Open in Browser
Local History
Synchronize 'window.ui'
Show in Explorer
File Path Ctrl+Alt+F12
Compare With... Ctrl+D
Compare File with Editor
External Tools
Diagrams
Create Gist...
QtDesigner
PyUIConv
Run
PyUIConv
Unregistered VCS root detected: The directory C:\Users\tank.sui\PycharmProjects is under Git, b

- démo




```
import sys
from PyQt5.QtWidgets import QApplication,QMainWindow
from window import Ui_MainWindow

if __name__ == '__main__':
    app = QApplication(sys.argv)
    w = QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(w)
    w.show()
    sys.exit(app.exec_())
```

Bonjour Monde Exemple

Cet exemple crée une fenêtre simple avec un bouton et une modification de ligne dans une mise en page. Il montre également comment connecter un signal à un emplacement, de sorte que

cliquer sur le bouton ajoute du texte à l'édition de ligne.

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


if __name__ == '__main__':

    app = QApplication(sys.argv)

    w = QWidget()
    w.resize(250, 150)
    w.move(300, 300)
    w.setWindowTitle('Hello World')
    w.show()

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

Une analyse

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

Chaque application PyQt5 doit créer un objet d'application. Le paramètre sys.argv est une liste d'arguments provenant d'une ligne de commande. Les scripts Python peuvent être exécutés à partir du shell.

```
w = QWidget()
```

Le widget `QWidget` est la classe de base de tous les objets d'interface utilisateur dans PyQt5. Nous fournissons le constructeur par défaut pour `QWidget` . Le constructeur par défaut n'a pas de parent. Un widget sans parent est appelé une fenêtre.

```
w.resize(250, 150)
```

La méthode `resize()` redimensionne le widget. Il est large de 250px et haut de 150px.

```
w.move(300, 300)
```

La méthode `move()` déplace le widget à une position sur l'écran à x = 300, y = 300 coordonnées.

```
w.setWindowTitle('Hello World')
```

Ici, nous définissons le titre de notre fenêtre. Le titre est affiché dans la barre de titre.

```
w.show()
```

La méthode `show()` affiche le widget à l'écran. Un widget est d'abord créé en mémoire et affiché ultérieurement à l'écran.

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

Enfin, nous entrons dans la boucle principale de l'application. La gestion des événements commence à partir de ce point. Le mainloop reçoit les événements du système de fenêtre et les distribue aux widgets d'application. Le mainloop se termine si nous appelons la méthode `exit()` ou si le widget principal est détruit. La méthode `sys.exit()` garantit une sortie propre. L'environnement sera informé de la fin de l'application.

La méthode `exec_()` a un trait de soulignement. C'est parce que exec est un mot clé Python. Et donc, `exec_()` été utilisé à la place.

Ajouter une icône d'application

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5.QtGui import QIcon


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        self.setGeometry(300, 300, 300, 220)
        self.setWindowTitle('Icon')
        self.setWindowIcon(QIcon('web.png'))

        self.show()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

Une analyse

Arguments de fonction en Python

En Python, les fonctions définies par l'utilisateur peuvent prendre quatre types d'arguments différents.

1. *Arguments par défaut:*

- Définition de fonction

  ```
  def defaultArg( name, msg = "Hello!"):
  ```

- Appel de fonction

  ```
  defaultArg( name)
  ```

2. *Arguments requis:*

- Définition de fonction

  ```
  def requiredArg (str,num):
  ```

- Appel de fonction:

  ```
  requiredArg ("Hello",12)
  ```

3. *Arguments de mots clés:*

- Définition de fonction

  ```
  def keywordArg( name, role ):
  ```

- Appel de fonction

  ```
  keywordArg( name = "Tom", role = "Manager")
  ```

  ou

  ```
  keywordArg( role = "Manager", name = "Tom")
  ```

4. *Nombre variable d'arguments:*

- Définition de fonction

  ```
  def varlengthArgs(*varargs):
  ```

- Appel de fonction

  ```
  varlengthArgs(30,40,50,60)
  ```

```
class Example(QWidget):

    def __init__(self):
        super().__init__()
        ...
```

Trois éléments importants de la programmation orientée objet sont les classes, les données et les méthodes. Ici, nous créons une nouvelle classe appelée `Example` . La classe `Example` hérite de la classe `QWidget` . Cela signifie que nous appelons deux constructeurs: le premier pour la classe `Example` et le second pour la classe héritée. La méthode `super()` renvoie l'objet parent de la classe `Example` et nous appelons son constructeur. La variable `self` réfère à l'objet lui-même.

Pourquoi avons-nous utilisé `__init__` ?

Regarde ça:

```
class A(object):
    def __init__(self):
        self.lst = []

class B(object):
```

```
    lst = []
```

et maintenant essayez:

```
>>> x = B()
>>> y = B()
>>> x.lst.append(1)
>>> y.lst.append(2)
>>> x.lst
[1, 2]
>>> x.lst is y.lst
True
```

et ça:

```
>>> x = A()
>>> y = A()
>>> x.lst.append(1)
>>> y.lst.append(2)
>>> x.lst
[1]
>>> x.lst is y.lst
False
```

> Est-ce que cela signifie que x dans la classe B est établi avant l'instanciation?

Oui, c'est un attribut de classe (il est partagé entre les instances). Alors que dans la classe A, c'est un attribut d'instance.

```
self.initUI()
```

La création de l'interface graphique est déléguée à la méthode `initUI()` .

```
self.setGeometry(300, 300, 300, 220)
self.setWindowTitle('Icon')
self.setWindowIcon(QIcon('web.png'))
```

Les trois méthodes ont été héritées de la classe `QWidget` . Le `setGeometry()` fait deux choses: il localise la fenêtre à l'écran et la définit. Les deux premiers paramètres sont les positions x et y de la fenêtre. La troisième est la largeur et la quatrième est la hauteur de la fenêtre. En fait, il combine les méthodes `resize()` et `move()` dans une méthode. La dernière méthode définit l'icône de l'application. Pour ce faire, nous avons créé un objet `QIcon` . Le `QIcon` reçoit le chemin d'accès à notre icône pour être affiché.

```
if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

L'application et les objets d'exemple sont créés. La boucle principale est démarrée.

Afficher une infobulle

```
import sys
from PyQt5.QtWidgets import (QWidget, QToolTip,
    QPushButton, QApplication)
from PyQt5.QtGui import QFont


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        QToolTip.setFont(QFont('SansSerif', 10))

        self.setToolTip('This is a <b>QWidget</b> widget')

        btn = QPushButton('Button', self)
        btn.setToolTip('This is a <b>QPushButton</b> widget')
        btn.resize(btn.sizeHint())
        btn.move(50, 50)

        self.setGeometry(300, 300, 300, 200)
        self.setWindowTitle('Tooltips')
        self.show()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

Une analyse

```
QToolTip.setFont(QFont('SansSerif', 10))
```

Cette méthode statique définit une police utilisée pour rendre les info-bulles. Nous utilisons une police 10px SansSerif.

```
self.setToolTip('This is a <b>QWidget</b> widget')
```

Pour créer une info-bulle, nous appelons la méthode `setTooltip()` . Nous pouvons utiliser le formatage de texte enrichi.

```
btn = QPushButton('Button', self)
btn.setToolTip('This is a <b>QPushButton</b> widget')
```

Nous créons un widget bouton-poussoir et créons une info-bulle pour cela.

```
btn.resize(btn.sizeHint())
btn.move(50, 50)
```

Le bouton est en cours de redimensionnement et déplacé sur la fenêtre. La méthode `sizeHint()` donne une taille recommandée pour le bouton.

Transformez votre projet en excutable / installateur

cx_Freeze - un outil peut emballer votre projet à excutable / installateur

- après l'installer par pip, pour emballer `demo.py` , nous avons besoin de `setup.py` ci-dessous.

```
import sys
from cx_Freeze import setup, Executable

# Dependencies are automatically detected, but it might need fine tuning.
build_exe_options = {
    "excludes": ["tkinter"],
    "include_files":[('./platforms','./platforms')] # need qwindows.dll for qt5 application
}

# GUI applications require a different base on Windows (the default is for a
# console application).
base = None
if sys.platform == "win32":
    base = "Win32GUI"

setup(  name = "demo",
        version = "0.1",
        description = "demo",
        options = {"build_exe": build_exe_options},
        executables = [Executable("demo.py", base=base)])
```

- puis construire

```
python .\setup.py build
```

- alors dist

```
python .\setup.py bdist_msi
```

Lire Démarrer avec pyqt5 en ligne: https://riptutorial.com/fr/pyqt5/topic/7403/demarrer-avec-pyqt5

Chapitre 2: Introduction aux barres de progression

Introduction

Les barres de progression font partie intégrante de l'expérience utilisateur et aident les utilisateurs à se faire une idée du temps qu'il reste pour un processus donné exécuté sur l'interface graphique. Cette rubrique passera en revue les bases de la mise en œuvre d'une barre de progression dans votre propre application.

Ce sujet abordera légèrement QThread et le nouveau mécanisme de signaux / slots. Des connaissances de base des widgets PyQt5 sont également attendues des lecteurs.

Lors de l'ajout d'exemples, utilisez uniquement les fonctions intégrées PyQt5 et Python pour démontrer la fonctionnalité.

PyQt5 seulement

Remarques

Expérimenter avec ces exemples est la meilleure façon de commencer à apprendre.

Examples

Barre de progression de base PyQt

C'est une barre de progression très basique qui n'utilise que ce qui est nécessaire au strict minimum.

Il serait sage de lire cet exemple jusqu'à la fin.

```
import sys
import time

from PyQt5.QtWidgets import (QApplication, QDialog,
                             QProgressBar, QPushButton)

TIME_LIMIT = 100

class Actions(QDialog):
    """
    Simple dialog that consists of a Progress Bar and a Button.
    Clicking on the button results in the start of a timer and
    updates the progress bar.
    """
    def __init__(self):
        super().__init__()
        self.initUI()
```

```
    def initUI(self):
        self.setWindowTitle('Progress Bar')
        self.progress = QProgressBar(self)
        self.progress.setGeometry(0, 0, 300, 25)
        self.progress.setMaximum(100)
        self.button = QPushButton('Start', self)
        self.button.move(0, 30)
        self.show()

        self.button.clicked.connect(self.onButtonClick)

    def onButtonClick(self):
        count = 0
        while count < TIME_LIMIT:
            count += 1
            time.sleep(1)
            self.progress.setValue(count)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = Actions()
    sys.exit(app.exec_())
```

La barre de progression est d'abord importée comme telle à `from PyQt5.QtWidgets import QProgressBar`

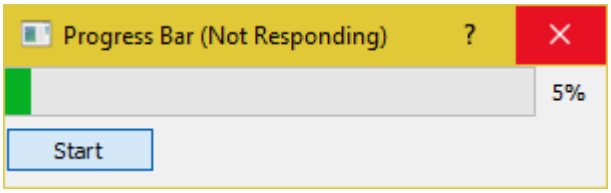
Ensuite, il est initialisé comme tout autre widget dans `QtWidgets`

La `self.progress.setGeometry(0, 0, 300, 25)` définit les positions `x,y` dans la boîte de dialogue et la largeur et la hauteur de la barre de progression.

Nous déplaçons ensuite le bouton en utilisant `.move()` de `30px` vers le bas pour qu'il y ait un écart de `5px` entre les deux widgets.

Ici, `self.progress.setValue(count)` est utilisé pour mettre à jour la progression. Définir une valeur maximale à l'aide de `.setMaximum()` calculera également automatiquement les valeurs pour vous. Par exemple, si la valeur maximale est définie sur 50, puisque `TIME_LIMIT` vaut 100, elle passera de 0 à 2 à 4% au lieu de 0 à 1 à 2 par seconde. Vous pouvez également définir une valeur minimale à l'aide de `.setMinimum()` ce qui oblige la barre de progression à partir d'une valeur donnée.

L'exécution de ce programme produira une interface graphique similaire à celle-ci.



Comme vous pouvez le constater, l'interface graphique `TIME_LIMIT` définitivement et ne répond plus jusqu'à ce que le compteur réponde à la condition `TIME_LIMIT` . En effet, `time.sleep` fait croire au système d'exploitation que ce programme est resté bloqué dans une boucle infinie.

QThread

Alors, comment pouvons-nous surmonter ce problème? Nous pouvons utiliser la classe de threading fournie par PyQt5.

```
import sys
import time

from PyQt5.QtCore import QThread, pyqtSignal
from PyQt5.QtWidgets import (QApplication, QDialog,
                             QProgressBar, QPushButton)

TIME_LIMIT = 100

class External(QThread):
    """
    Runs a counter thread.
    """
    countChanged = pyqtSignal(int)

    def run(self):
        count = 0
        while count < TIME_LIMIT:
            count +=1
            time.sleep(1)
            self.countChanged.emit(count)

class Actions(QDialog):
    """
    Simple dialog that consists of a Progress Bar and a Button.
    Clicking on the button results in the start of a timer and
    updates the progress bar.
    """
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle('Progress Bar')
        self.progress = QProgressBar(self)
        self.progress.setGeometry(0, 0, 300, 25)
        self.progress.setMaximum(100)
        self.button = QPushButton('Start', self)
        self.button.move(0, 30)
        self.show()

        self.button.clicked.connect(self.onButtonClick)

    def onButtonClick(self):
        self.calc = External()
        self.calc.countChanged.connect(self.onCountChanged)
        self.calc.start()

    def onCountChanged(self, value):
        self.progress.setValue(value)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = Actions()
    sys.exit(app.exec_())
```

Décomposons ces modifications.

```
from PyQt5.QtCore import QThread, pyqtSignal
```

Cette ligne importe `Qthread` qui est une implémentation `PyQt5` pour diviser et exécuter certaines parties (par exemple: fonctions, classes) d'un programme en arrière-plan (également appelé multi-threading). Ces parties sont également appelées threads. Tous les programmes `PyQt5` par défaut ont un thread principal et les autres (threads de travail) sont utilisés pour décharger en arrière-plan des tâches intensives et traiter des tâches intensives tout en conservant le fonctionnement du programme principal.

Le second import `pyqtSignal` est utilisé pour envoyer des données (signaux) entre les threads de travail et principaux. Dans cette instance, nous l'utilisons pour indiquer au thread principal de mettre à jour la barre de progression.

Nous avons maintenant déplacé la boucle while pour le compteur dans une classe distincte appelée `External` .

```
class External(QThread):
    """
    Runs a counter thread.
    """
    countChanged = pyqtSignal(int)

    def run(self):
        count = 0
        while count < TIME_LIMIT:
            count +=1
            time.sleep(1)
            self.countChanged.emit(count)
```

En sous-classant `QThread` nous convertissons essentiellement `External` en une classe qui peut être exécutée dans un thread distinct. Les threads peuvent également être démarrés ou arrêtés à tout moment, ce qui ajoute à ses avantages.

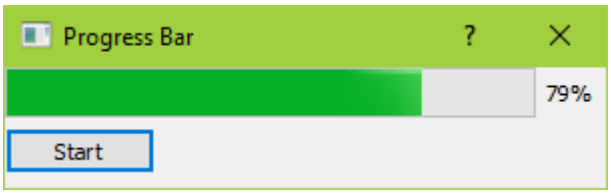
Ici `countChanged` est la progression en cours et `pyqtSignal(int)` indique au thread de travail que le signal envoyé est de type `int` . Bien que `self.countChanged.emit(count)` envoie simplement le signal à toutes les connexions du thread principal (normalement, il peut également être utilisé pour communiquer avec d'autres threads de travail).

```
def onButtonClick(self):
        self.calc = External()
        self.calc.countChanged.connect(self.onCountChanged)
        self.calc.start()

def onCountChanged(self, value):
    self.progress.setValue(value)
```

Lorsque l'utilisateur clique sur le bouton, `self.onButtonClick` s'exécute et démarre également le thread. Le thread est démarré avec `.start()` . Il convient également de noter que nous avons connecté le signal `self.calc.countChanged` créé précédemment à la méthode utilisée pour mettre à jour la valeur de la barre de progression. À chaque mise à jour de `External::run::count` la valeur `int` est également envoyée à `onCountChanged` .

Voici comment l’interface graphique peut s’occuper de ces modifications.



Il devrait également se sentir beaucoup plus réactif et ne gèlera pas.

Lire Introduction aux barres de progression en ligne:
https://riptutorial.com/fr/pyqt5/topic/9544/introduction-aux-barres-de-progression

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec pyqt5	Ansh Kumar, Community, ekhumoro, suiwenfeng
2	Introduction aux barres de progression	daegontaven