Spécialité NSI	Création de programme fenêtré avec pyQt	1ere

Objectif de l'activité :

- Créer des programme python avec une interface graphique
- Utiliser une documentation
- Manipuler des fonctions et des objets

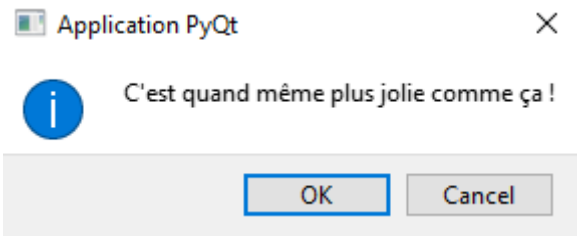
Prérequis :

- Savoir prototyper une fonction
- Savoir manipuler les fonctions
- Avoir de bonnes bases en python
- Logiciel pyCharm installé, bibliothèques Qt disponible.

Remarques :

- L'activité est trop longue, supprimer la partie sur le designer correspond mieux à un format de 2h. Le mini-projet est une idée à retravailler mais qui peut donner du grain aux élèves très en avance.

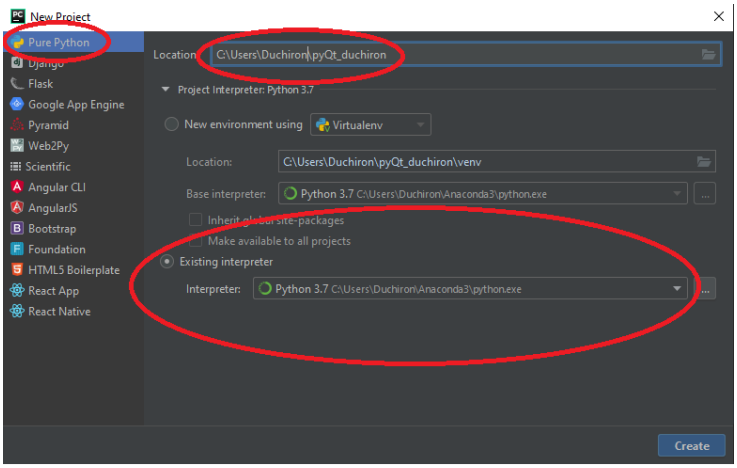
Mise en situation : Vous connaissez maintenant les bases du langage python, cependant pour l'instant nous avons toujours affiché tous nos résultats dans la console de pyCharm. Hors lorsque l'on développe un programme complet, on préfère généralement avoir un résultat un peu plus joli et surtout plus intuitif dans l'utilisation.
PyQt est un ensemble de bibliothèques qui permet cela, et vous allez apprendre à l'utiliser !



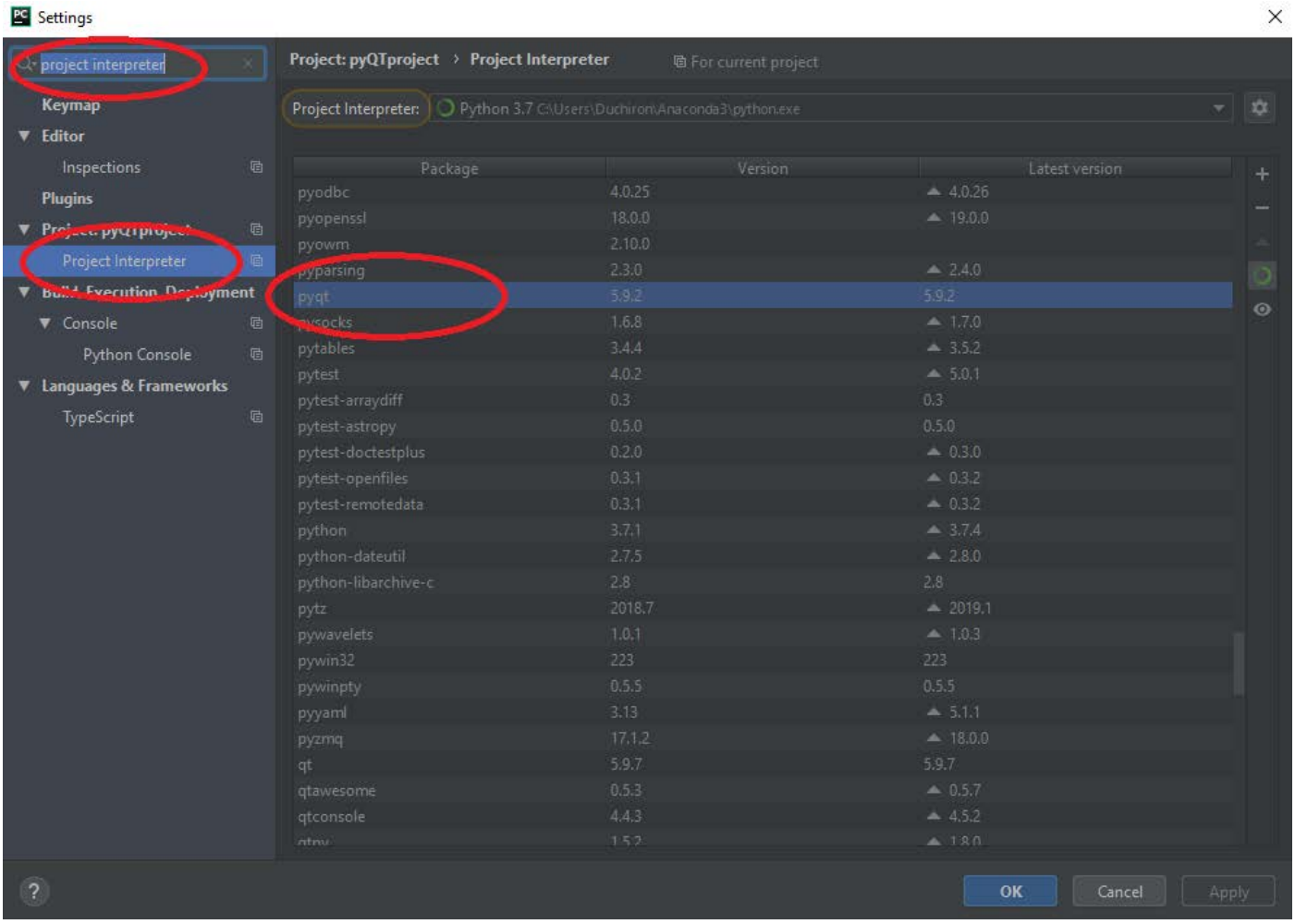
1. Installation des éléments nécessaires :

PyQt contient des milliers de bibliothèques, avant toute chose il faut s'assurer que pyCharm sait où les trouver.

- Créer un nouveau projet intitulé ; pyQt_nom
- Dans l'interface de création du projet : Créer le projet comme vous l'avez appris en respectant les champs suivants :



- Ensuite dans le menu file/settings :
  - Rechercher : project interpreter, vous devriez arriver sur la fenêtre suivante dans laquelle vous allez chercher pyQt, s'il n'y est pas, cliquer sur le + à droite pour l'ajouter !



Voilà, tout est maintenant fonctionnel pour utiliser les bibliothèques de pyQt.

2. Ma première fenêtre :

Copier le code suivant et exécuter le :

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


def applicationpyQt():
    app = QApplication(sys.argv)
    windows = QWidget()
    windows.show()
    sys.exit(app.exec_())


if __name__ == '__main__':
    applicationpyQt()
```

Analysons ce code :

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

Cette première partie contient l'importation de toutes les bibliothèques nécessaires pour faire une simple fenêtre avec Qt

```
def applicationpyQt():
    app = QApplication(sys.argv)
    windows = QWidget()
    windows.show()
    sys.exit(app.exec_())
```

Cette deuxième partie est un peu plus complexe, il s'agit de la fonction qui va créer et gérer la fenêtre.

`app = QApplication(sys.argv)` : Crée l'application
`windows = QWidget()` : Crée l'objet qui servira de conteneur pour les autres éléments
`windows.show()` : Affiche l'objet principal
`sys.exit(app.exec_())` : Exécute l'application
Bon c'est bien, mais ça manque un peu de contenu !

Ajouter la ligne :

`windows.setWindowTitle('PyQt')` juste après la création de l'objet windows

Q1 : Observer la fenêtre, quel est le changement ? Le titre de la fenêtre a changé.

3. La documentation Qt :

Qt est ce qu'on appelle un framework, c'est-à-dire qu'il s'agit d'un ensemble de milliers de bibliothèques développées pour faire des applications complètes. Pour utiliser ce framework il existe une documentation officielle de Qt qui permet de savoir comment utiliser les différentes fonctions.

Ajouter la ligne : `windows.setGeometry(200, 200, 200, 200)` à la suite du dernier ajout.

Q2. Quel est le changement ? La dimension de la fenêtre a changé.

D'après les explications précédentes, pourquoi faut-il toujours laisser les lignes windows.show() et sys.exit(app.exec_()) à la fin de la fonction ?

Parce que si cela reste avant les nouvelles lignes, l'affichage se fera avant que ces lignes soient exécutées et elles ne seront donc pas prises en compte.

La documentation de la méthode setGeometry est décrite ici :
https://doc.qt.io/qtforpython/PySide2/QtWidgets/QWidget.html#PySide2.QtWidgets.PySide2.QtWidgets.QWidget.setGeometry

Exercice 1 : En utilisant cette documentation, changer les dimensions de la fenêtre de façon à obtenir une fenêtre de 500 pixels de large, 300 pixels de haut, placer à 50 pixels du haut et à 50 pixels de la gauche de l'écran.
(N.B. : W signifie généralement width, et H signifie généralement Height)

De façon général, l'ensemble de la documentation Qt est disponible ici :

https://doc.qt.io/qtforpython/modules.html

Désormais lorsque vous cherchez la façon d'utiliser un objet Qt, vous devez vous référer à cette documentation.

4. Ajoutons du texte :

Remplissons un peu la fenêtre avec du texte, pour cela on utilise l'objet QLabel du module QWidgets.

Voici la ligne à ajouter pour créer un label de texte vide :

```
text = QtWidgets.QLabel(windows)
```

Explication : on nomme le Label « text », on le déclare comme QLabel, on lui donne comme paramètre l'objet 'windows' qui était notre fenêtre. C'est cette affectation qui permet de faire en sorte que le Label soit dans la fenêtre.

Exercice 2 :

a. A partir de la documentation du QLabel (disponible ici), faites-en sorte d'ajouter le texte : « Vive la NSI » dans le label.
b. Déplacer le label pour qu'il soit à peu prêt au centre de la fenêtre
c. Modifier maintenant votre programme pour que le Label affiche le résultat de la multiplication entre 2 variables contenant des nombres de votre choix.

5. Ajoutons des boutons :

Nous progressons vers une véritable application, maintenant il est temps d'ajouter des boutons !

Voici le prototype pour la création d'un bouton :

```
bouton = QtWidgets.QPushButton('texte', conteneur)
```

Exercice 3 :

- Créer un bouton avec le texte 'Go' et qui est contenu dans la fenêtre
- Déplacer le pour qu'il soit juste sous le label

Cours : Les signaux et les slots :

Avoir un bouton c'est bien mais s'il ne fait rien, ça n'est pas très utile. Pour cela il est essentiel de comprendre un des fonctionnements de Qt, les signaux et les slots :

Les signaux : *Certains objets de Qt peuvent émettre des signaux dans certains cas bien précis qui sont préprogrammés dans les bibliothèques de Qt. Ces signaux sont des événements cela permet donc de prévenir le reste du programme que tel ou tel événement s'est produit. Voici une liste de quelques signaux usuels :*

- *Clicked : est cliqué*
- *Activated : est activé*
- *Etc.*

Chaque objet possède potentiellement des signaux différents, en effet il paraît logique qu'on n'obtienne pas les mêmes usages entre un bouton, une zone de texte ou une liste déroulante.

Pour connaître les signaux disponibles sur un objet, il faut se référer à la documentation de Qt.

Exercice 4 : *A partir de la documentation sur les boutons (ici), identifier les signaux et donner l'événement auxquels ils correspondent :*

Clicked : détecte que le bouton a été cliqué

Pressed : détecte que le bouton est appuyé

Released : détecte quand le bouton est relâché

Toggled : détecte si le bouton est coché (pour les boutons cochable)

Les slots : *Le slot c'est une fonction qui sera exécutée lorsque le signal sera reçu. Par conséquent, c'est dans cette fonction que l'on va placer le code que l'on souhaite exécuter lors de la réception du signal. Un signal une fois émit doit toujours être connecté sur un slot.*

Il existe certains slots qui sont préprogrammés et utilisable directement, mais la plupart du temps il faudra coder soi-même la fonction qui va servir de slot.

Exemple : voici la ligne qui permet de connecter un bouton à un slot :

```
button.clicked.connect(bouton_go)
```

Détaillons la ligne :

- *button : C'est le nom de mon objet bouton que j'ai créé avant dans mon code.*
- *clicked : C'est le signal que je connecte, ici le signal sera émit lorsque l'utilisateur cliquera sur le bouton*
- *connect : pour indiquer sur quoi on connecte le signal*
- *bouton_go : C'est le nom de la fonction qui va être exécuté lorsque le signal sera reçu.*

Exercice 5-1 : *Essayer d'ajouter la ligne précédente à votre code (en la modifiant pour que cela corresponde à votre bouton), quel est le problème rencontré ?*

La fonction bouton_go n'est pas référencé et le code ne peut pas s'exécuter.

Exercice 5-2 : *Ajouter la fonction bouton_go à votre code et faites-en sorte qu'elle affiche 'Bonjour' dans le Label.*

6. Récupérer du texte entrer par l'utilisateur :

Pour avoir un programme véritablement interactif, il faut que l'utilisateur puisse entrer lui-même des informations, et particulièrement du texte.

Un des objets qui permet cela est le QLineEdit, la façon de le créer est la même que celle du bouton.

Ensuite la méthode pour récupérer le texte s'appelle text().

Exercice 6 : A partir de la documentation et des signaux de QLineEdit :

- Créer un objet QLineEdit sous le bouton.
- Créer le slot text_change qui va afficher dans le label la valeur contenue dans le QLineEdit.
- Connecter le au bon signal de l'objet pour que le contenu du label change lorsque le QLineEdit est modifié.

7. QtDesigner :

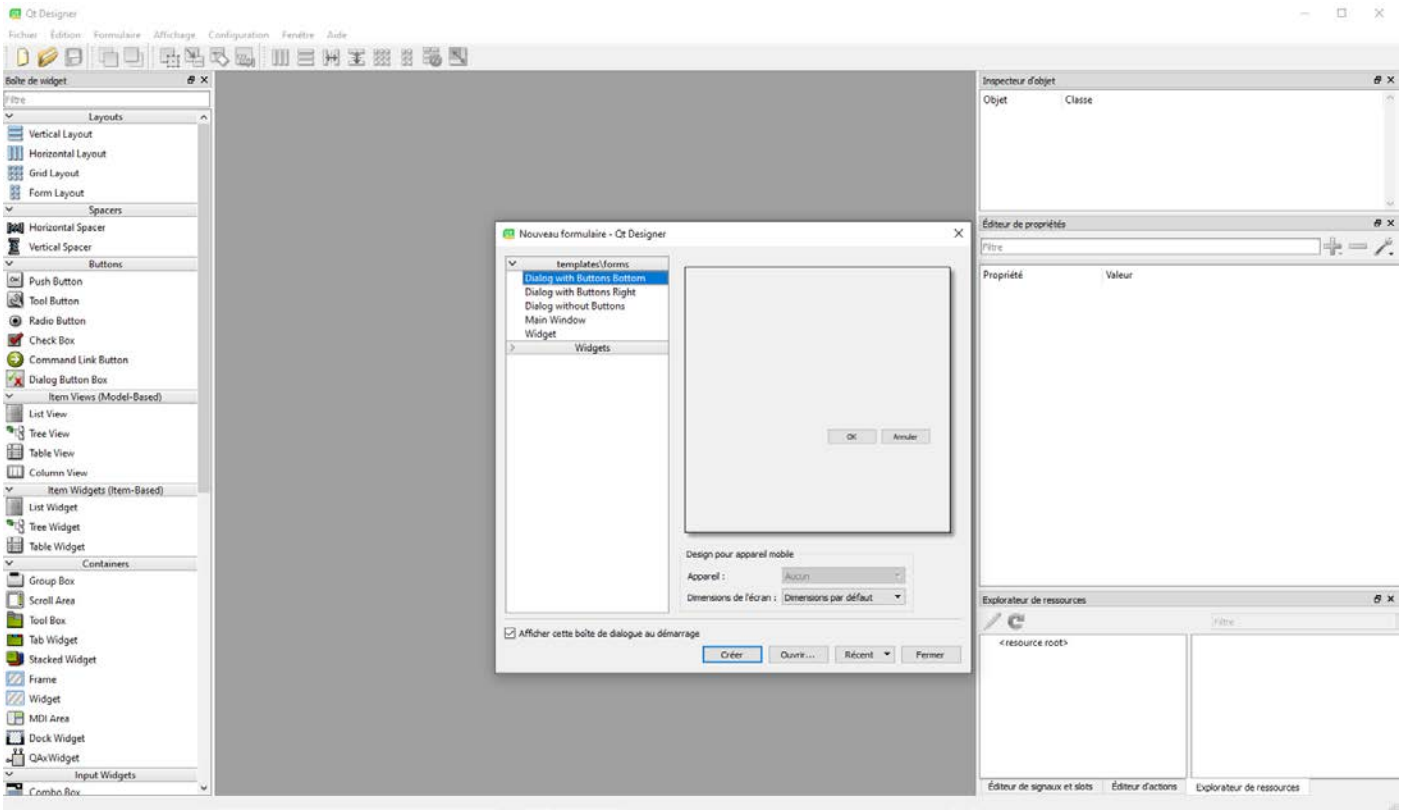
Si vous avez pris la peine de jeter un coup d'œil à la documentation de Qt vous avez vu qu'il existe des centaines et des centaines d'objet graphiques. Il est compliqué de tous les connaître et de savoir lequel utiliser.

De plus il n'est pas aisé de placer les éléments exactement comme on le souhaite.

Heureusement il existe une interface graphique pour créer l'interface graphique ! Elle s'appelle QtDesigner.

Pour la lancer taper : Designer dans le terminal de pyCharm de votre projet.

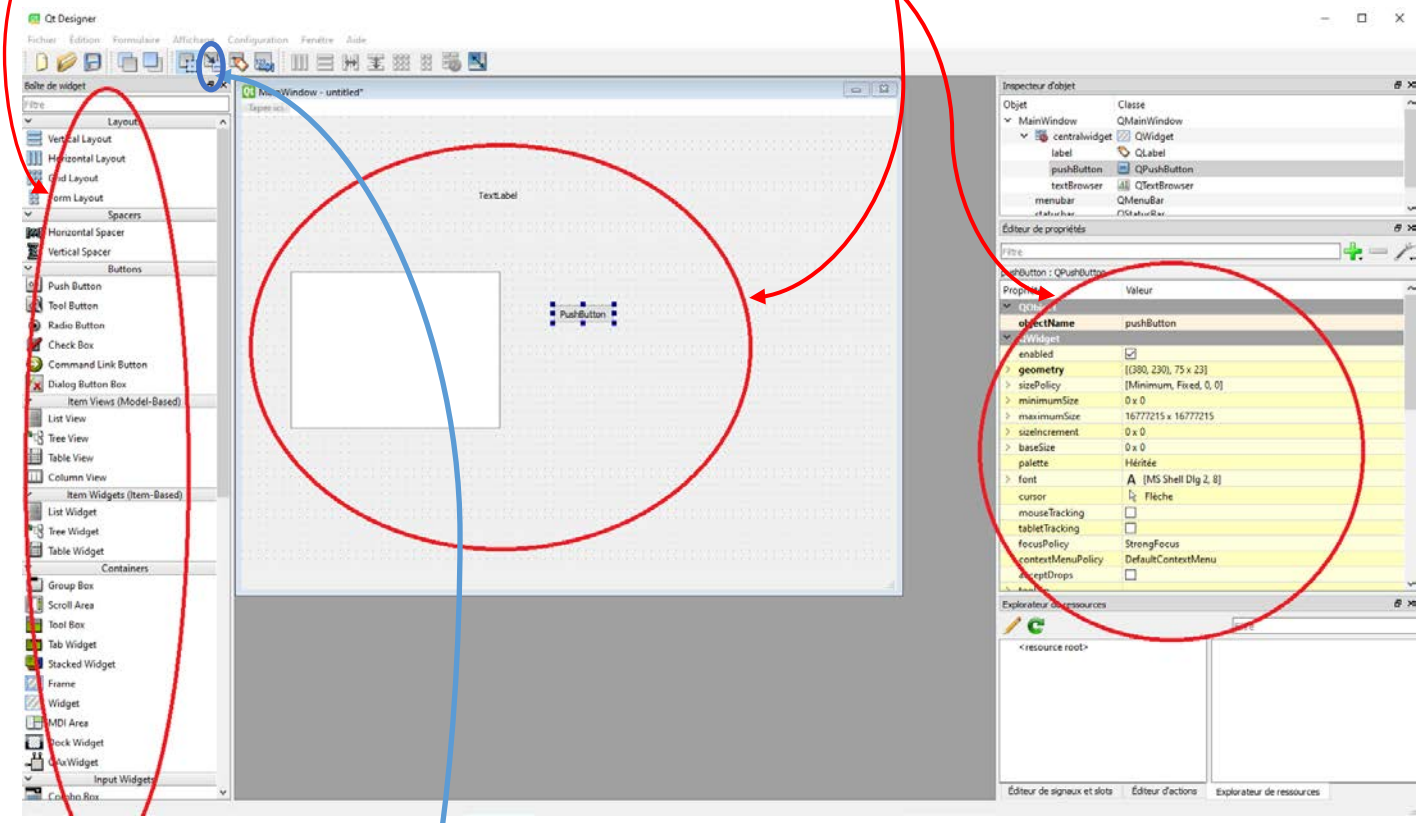
Vous devriez obtenir une fenêtre semblable à ça :



- Sélectionnez Main windows puis cliquez sur créer

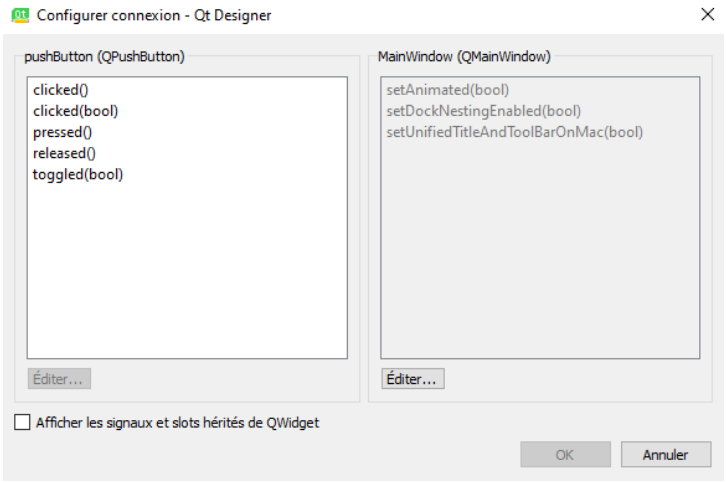
Vous arrivez dans la fenêtre de création.

- Au centre ce à quoi ressemble votre application
- A gauche tous les objets graphiques disponible
- A droite les propriétés de vos objets graphiques



Une fois que vous ajouter tous les éléments que vous voulez, vous pouvez configurez les signaux et les slots, ici :

- Pour éditer les signaux et slots, sélectionnez un élément et glisser dans le vide, la fenêtre suivante va s'ouvrir :



- A gauche vous avez les signaux disponibles pour l'objet, sélectionnez celui que vous voulez.

- A droite cliquez sur Editer… puis sur le + dans la rubrique slot
- Ajouter le slot que vous voulez

Une fois que vous avez configurer la partie graphique et les comportements, vous n'avez plus qu'à sauvegarder votre fichier au format .ui dans votre dossier de projet.

Il ne reste plus qu'à faire le lien avec python :

Dans le terminal de pyCharm exécuter la ligne de commande :

```
pyuic5 nomFichier.ui -o nomFichier.py
```

Cela a généré un programme python.

- Ouvrez le fichier python designer1.py
- Modifier la ligne 2 pour remplacer from debut, par from nom_de_votre_fichier
- Modifier la fonction bouton_go pour qu'elle corresponde à votre slot.
- Modifier la fonction text_change pour qu'elle corresponde à votre slot.
- En cas de problème essayer d'importer le fichier monFichier.py dans votre projet et regardez les différences avec le vôtre !

8. Mini-Projet :

Vous allez maintenant développer une application complète pour vous entraîner.

Nous vous suggérons de créer une calculatrice 4 opérations pour commencer.

Vous pouvez solliciter votre professeur si vous avez votre propre idée d'application.