La gestion des boites de dialogue et des interfaces

Les codes Python (les « algorithmes ») que l'on produit ont principalement deux finalités. Soit les codes sont construits pour nous-même (voire quelques personnes compétentes), auquel cas de simples commentaires peuvent permettre de réutiliser simplement le code ou de l'adapter à de nouvelles données. Soit les codes produits doivent pouvoir servir à des utilisateurs quelconques, auquel cas il faut chercher à interagir avec ces utilisateurs pour exécuter les algorithmes. Cette deuxième finalité implique donc un travail supplémentaire. Le moyen le plus simple pour interagir avec un utilisateur est d'utiliser des boites de dialogue classiques, mais très vite si le code requiert de nombreuses informations pour fonctionner, ou permet de nombreuses personnalisations, il faudra créer des boites de dialogue complexes que l'on peut appeler « interface ». Ce que l'on peut appeler l' « interfaçage », qui va donc mettre en relation l'utilisateur avec nos algorithmes, est une étape qui peut se révéler très chronophage. On peut ainsi passer plus de temps à créer ces interfaces qu'à produire les algorithmes nécessaires à la résolution du problème posé. Cela est principalement dû au fait que l'interfaçage nécessite en théorie de trouver des réponses pour chaque cas d'utilisation de l'utilisateur. ***Dans ce TD, on va partir d'un algorithme simple qui effectue une intersection entre deux couches d'information géographique. On va alors créer des boites de dialogue et des interfaces en partant du plus simple pour aller vers le plus complexe*** (qui sera bien souvent le plus pratique pour l'utilisateur)**.** Les données utilisées sont disponibles ici : http://sergelhomme.fr/data/Donnees.zip . On charge dans QGIS uniquement les Shapefiles des départements et des magasins Leclerc. Puis, on va créer des buffers de 30km autour des magasins. On appellera cette couche « Tampon ».

1) Les boites de dialogues : la base de l'interfaçage

On va partir d'un code (algorithme) simple qui permet d'intersecter nos deux couches, celle des départements et celle des buffers (que l'on a créé spécialement pour le TD et qui peut très bien être en mémoire).

```
param = {'INPUT': 'Departements',
         'OVERLAY': 'Tampon',
         'OUTPUT':  'memory:'
       }
intersection = processing.run('native:intersection', param)
QgsProject.instance().addMapLayer(intersection['OUTPUT'])
intersection['OUTPUT'].setName('Intersection')
```

Le moyen le plus simple pour communiquer avec l'utilisateur est sans doute de lui demander d'écrire le nom des couches. Pour cela, on peut utiliser la boite de dialogue « getText() » qui appartient à la méthode « QInputDialog ». Ici, on peut donc créer deux boites de dialogue de ce type pour récupérer le nom de deux couches.

```
nomZone = QInputDialog.getText(None,"Mag","Entrez le nom de la couche Zone :")
nomTampon = QInputDialog.getText(None,"Zone","Entrez le nom de la couche Tampon :")
```

Pour utiliser les noms entrés par l'utilisateur, il faut les réinjecter dans le code précédent. Attention, la méthode « getText() » renvoie une liste et non juste la valeur (le nom).

```
param = {'INPUT': nomZone[0],
         'OVERLAY': nomTampon[0],
        'OUTPUT':  'memory:'
       }
intersection = processing.run('native:intersection', param)
QgsProject.instance().addMapLayer(intersection['OUTPUT'])
intersection['OUTPUT'].setName('Intersection')
```

Ecrire les noms des couches n'est pas très pratique pour l'utilisateur, on peut alors envisager lui demander le numéro de la couche avec la méthode « getInt() ». Ce sera a prioi plus simple.

```
numDep = QInputDialog.getInt(None,"N°","Entrez le numero de la couche Zone")
numTampon = QInputDialog.getInt(None,"N°","Entrez le numero de la couche")
```

Néanmoins, pour nous, ça complique l'exécution de l'algorithme. En effet, il faut alors récupérer le nom de la couche (ou la couche) à partir de l'index rentré par l'utilisateur. De surcroit, le système d'index de Python commence à zéro alors que les utilisateurs vont commencer leur compte par 1. On part aussi du principe qu'ils partent du haut vers le bas pour compter (on peut le préciser dans les boites de dialogue sinon).

```
coucheZone = qgis.utils.iface.mapCanvas().layers()[numDep[0]-1]
coucheTampon = qgis.utils.iface.mapCanvas().layers()[numTampon[0]-1]
param = {'INPUT': coucheZone,
         'OVERLAY': coucheTampon,
        'OUTPUT':  'memory:'
       }
intersection = processing.run('native:intersection', param)
QgsProject.instance().addMapLayer(intersection['OUTPUT'])
intersection['OUTPUT'].setName('Intersection')
```

Cependant, la méthode la simple pour l'utilisateur dans ce contexte est de lui proposer des listes déroulantes lui permettant de sélectionner ses couches. Pour cela, on dispose de la méthode getItem().

```
QInputDialog.getItem(None, "Truc", "Quel truc choisir ?",[ 'Truc1','Truc2'])
```

Reste pour utiliser cette méthode, à récupérer une liste des noms des couches pour les soumettre à l'utilisateur dans les deux boites de dialogue.

```
couches = qgis.utils.iface.mapCanvas().layers()
nomCouches=[]
for i in range(len(couches)) :
      nomCouches = nomCouches + [couches[i].name()]

nomZone = QInputDialog.getItem(None, "Couches", "Couche Zone",nomCouches)
nomTampon = QInputDialog.getItem(None, "Couches", "Couche Tampon",nomCouches)
param = {'INPUT': nomZone[0],
         'OVERLAY': nomTampon[0],
        'OUTPUT':  'memory:'
       }
intersection = processing.run('native:intersection', param)
QgsProject.instance().addMapLayer(intersection['OUTPUT'])
intersection['OUTPUT'].setName('Intersection')
```

A noter que les boites dialogues marchent dans les deux sens et que l'on peut s'en servir pour envoyer des messages à l'utilisateur, comme un message de fin par exemple ou un résultat numérique.

```
QMessageBox.information(None, "Information", "Calcul fini")
```

2) Développer une interface : les boites de dialogue complexes

La limite de cette approche par boites de dialogue successives, c'est que lorsque l'on a besoin de beaucoup d'informations, cela devient relativement lourd et cela demande même parfois à l'utilisateur de se souvenir de ces anciens choix. C'est pour cela que les logiciels privilégient généralement des interfaces (des boites de dialogue plus complexes) qui synthétisent ce que l'on pourrait faire à l'aide de plusieurs boites de dialogue. Avec PyQt, il est possible de développer des interfaces en Python. Pour créer une interface, c'est d'une simplicité sans nom. Il suffit d'utiliser la méthode QWidget() puis de l'afficher avec la méthode show().

```
Dialog = QWidget()
```

```
Dialog.show()
```

C'est rudimentaire, mais c'est déjà pas si mal… On va ensuite pouvoir personnaliser cette interface en la redimensionnant et en rajoutant des boutons classiques d'exécution. Pour cela, on va utiliser des objets (bibliothèques) de PyQT qu'il faut d'importer.

```
from PyQt5 import QtCore, QtGui, QtWidgets
Dialog.resize(400, 200)
buttonBox = QtWidgets.QDialogButtonBox(Dialog)
buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogBu
ttonBox.Ok)
Dialog.show()
```

On peut ensuite positionner les objets créer avec la méthode « setGeometry() ».

```
buttonBox.setGeometry(QtCore.QRect(50, 140, 300, 30))
```

On pourra écrire dans l'interface à l'aide de « label ».

```
label = QtWidgets.QLabel(Dialog)
label.setGeometry(QtCore.QRect(50, 30, 150, 25))
label.setText("Couche Zone")
```

Ensuite, pour que l'utilisateur puisse renseigner ces informations, il pourra utiliser les mêmes outils que pour les boites de dialogue (zone de texte, liste déroulante, case à cocher…). Pour que l'utilisateur écrive des lignes de texte, il est possible d'utiliser des « LineEdit() ».

```
textEdit = QtWidgets.QLineEdit(Dialog)
textEdit.setGeometry(QtCore.QRect(220, 30, 130, 25))
```

Pour les listes déroulantes, on pourra utiliser des « comboBox() ».

```
comboBox = QtWidgets.QComboBox(Dialog)
comboBox.setGeometry(QtCore.QRect(220, 80, 130, 25))
```

L'enjeu des listes déroulantes est alors de leur attribuer des valeurs.

```
comboBox.addItems(["fsg","gfd"])
```

L'enjeu pour nous sera de rajouter des informations à l'aide de PyQGIS, comme le nom des couches, comme précédemment en fait…

```
couches = qgis.utils.iface.mapCanvas().layers()
nomCouches=[]
for i in range(len(couches)) :
      nomCouches = nomCouches + [couches[i].name()]
comboBox.addItems(nomCouches)
```

On peut alors obtenir le code suivant pour dessiner notre interface :

```
from PyQt5 import QtCore, QtGui, QtWidgets
couches = qgis.utils.iface.mapCanvas().layers()
nomCouches=[]
for i in range(len(couches)) :
      nomCouches = nomCouches + [couches[i].name()]

Dialog = QWidget()
```

```
Dialog.resize(400, 200)
buttonBox = QtWidgets.QDialogButtonBox(Dialog)
buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogBu
ttonBox.Ok)
buttonBox.setGeometry(QtCore.QRect(50, 140, 300, 30))
label = QtWidgets.QLabel(Dialog)
label.setGeometry(QtCore.QRect(50, 30, 150, 25))
label.setText("Couche Zone")
label2 = QtWidgets.QLabel(Dialog)
label2.setGeometry(QtCore.QRect(50, 80, 150, 25))
label2.setText("Couche Tampon")
textEdit = QtWidgets.QLineEdit(Dialog)
textEdit.setGeometry(QtCore.QRect(220, 30, 130, 25))
comboBox = QtWidgets.QComboBox(Dialog)
comboBox.setGeometry(QtCore.QRect(220, 80, 130, 25))
comboBox.addItems(nomCouches)
Dialog.show()
```

Il ne reste plus qu'à interagir avec les boutons pour lancer le calcul (l'algorithme). Pour cela, on va définir une fonction qui exécute ce que l'on souhaite faire quand la personne clique sur « ok ». On va appeler cette fonction « appui_ok ». Il faut la placer en amont de notre interface.

```
def appui_ok() :
    print("Hello")
```

Pour exécuter cette fonction, il faut indiquer que le bouton « OK » lance cette fonction « appui_ok ».

```
buttonBox.accepted.connect(appui_ok)
```

Il est possible de fermer la boite de dialogue à l'aide de la méthode « close() ».

```
Dialog.close()
```

On pourra alors fermer l'interface une fois le calcul terminé, mais aussi fermer l'interface lorsque l'utilisateur clique sur « Annuler » en rajouter la commande au suivante au boutonbox et en créant la fonction « appui_no ».

```
buttonBox.rejected.connect(appui_no)
```

On pourra récupérer les informations remplies par l'utilisateur à l'aide des commandes suivantes :

```
print(textEdit.text())
print(comboBox.currentText())
```

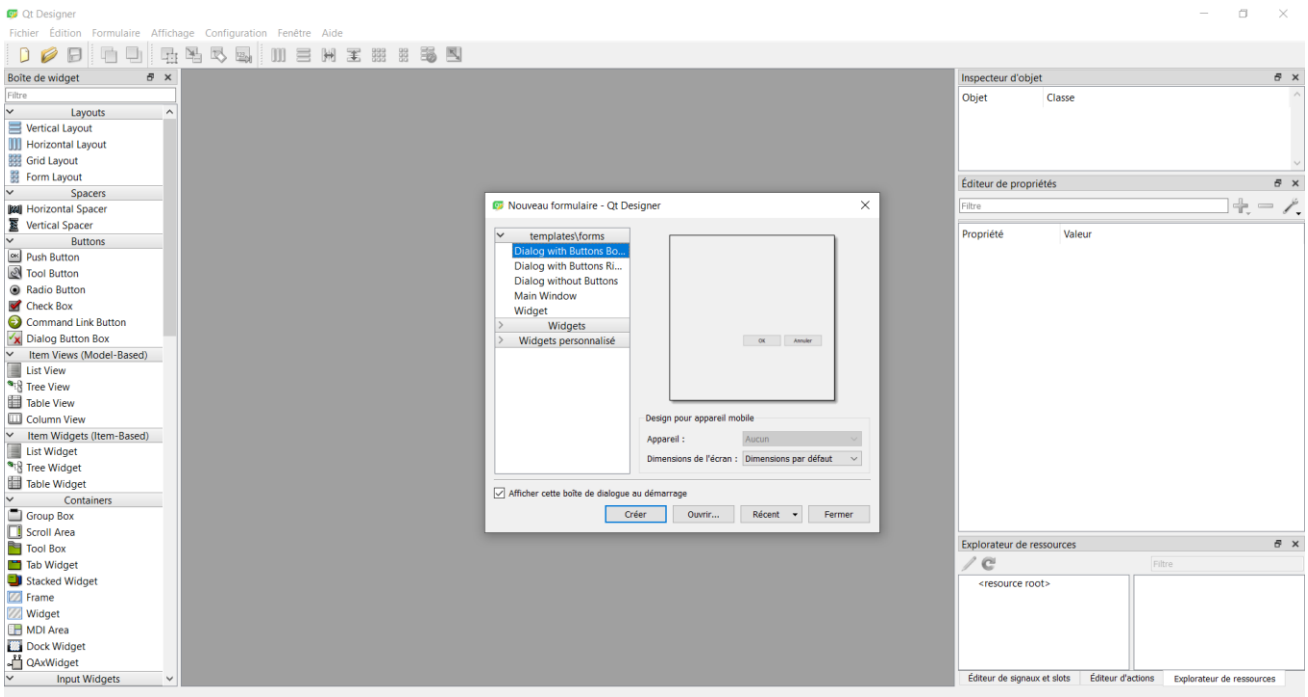
On peut alors utiliser les informations de l'utilisateur pour exécuter notre intersection.

```
from PyQt5 import QtCore, QtGui, QtWidgets
def appui_ok() :
    nomZone = textEdit.text()
    nomTampon = comboBox.currentText()
    param = {'INPUT': nomZone,
         'OVERLAY': nomTampon,
         'OUTPUT':  'memory:'
         }
    intersection = processing.run('native:intersection', param)
    QgsProject.instance().addMapLayer(intersection['OUTPUT'])
    intersection['OUTPUT'].setName('Intersection')
    Dialog.close()
```

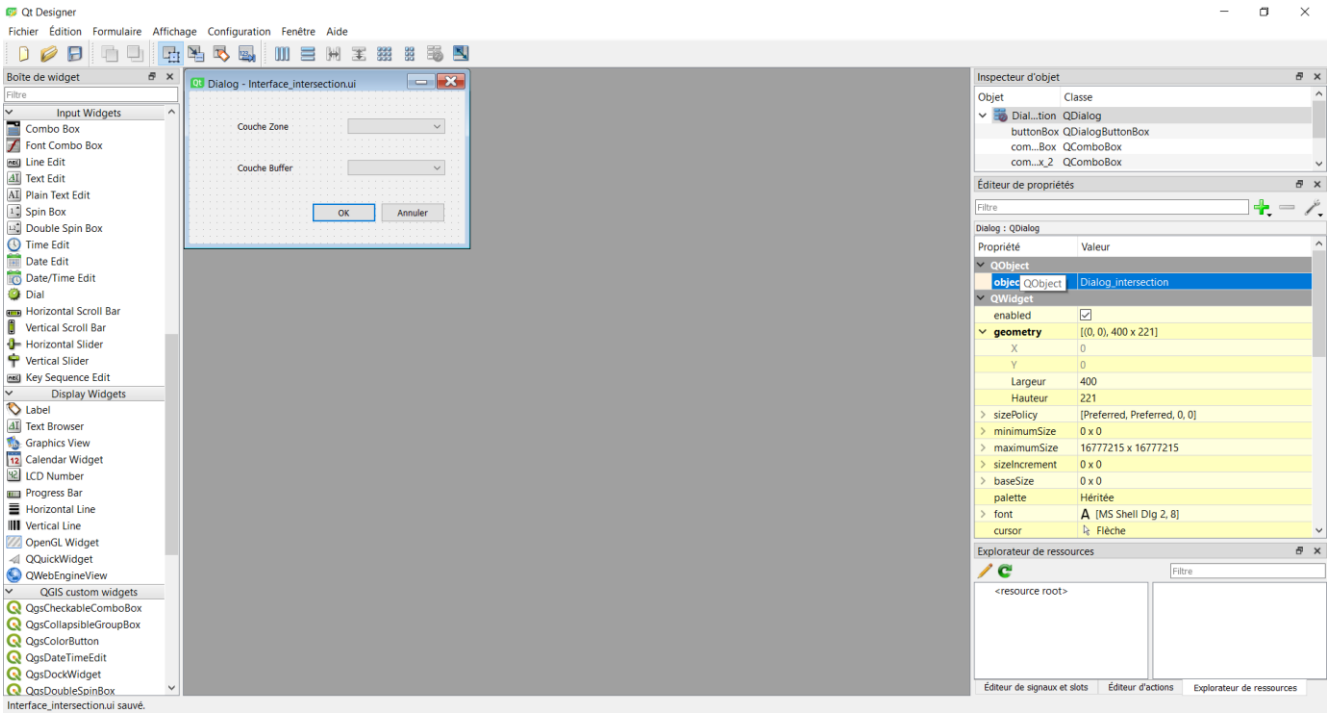
```
def appui_no() :
    Dialog.close()

couches = qgis.utils.iface.mapCanvas().layers()
nomCouches=[]
for i in range(len(couches)) :
      nomCouches = nomCouches + [couches[i].name()]

Dialog = QWidget()
Dialog.resize(400, 200)
buttonBox = QtWidgets.QDialogButtonBox(Dialog)
buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogBu
ttonBox.Ok)
buttonBox.setGeometry(QtCore.QRect(50, 140, 300, 30))
buttonBox.accepted.connect(appui_ok)
buttonBox.rejected.connect(appui_no)
label = QtWidgets.QLabel(Dialog)
label.setGeometry(QtCore.QRect(50, 30, 150, 25))
label.setText("Couche Zone")
label2 = QtWidgets.QLabel(Dialog)
label2.setGeometry(QtCore.QRect(50, 80, 150, 25))
label2.setText("Couche Tampon")
textEdit = QtWidgets.QLineEdit(Dialog)
textEdit.setGeometry(QtCore.QRect(220, 30, 130, 25))
comboBox = QtWidgets.QComboBox(Dialog)
comboBox.setGeometry(QtCore.QRect(220, 80, 130, 25))
comboBox.addItems(nomCouches)
Dialog.show()
```

3) QT designer

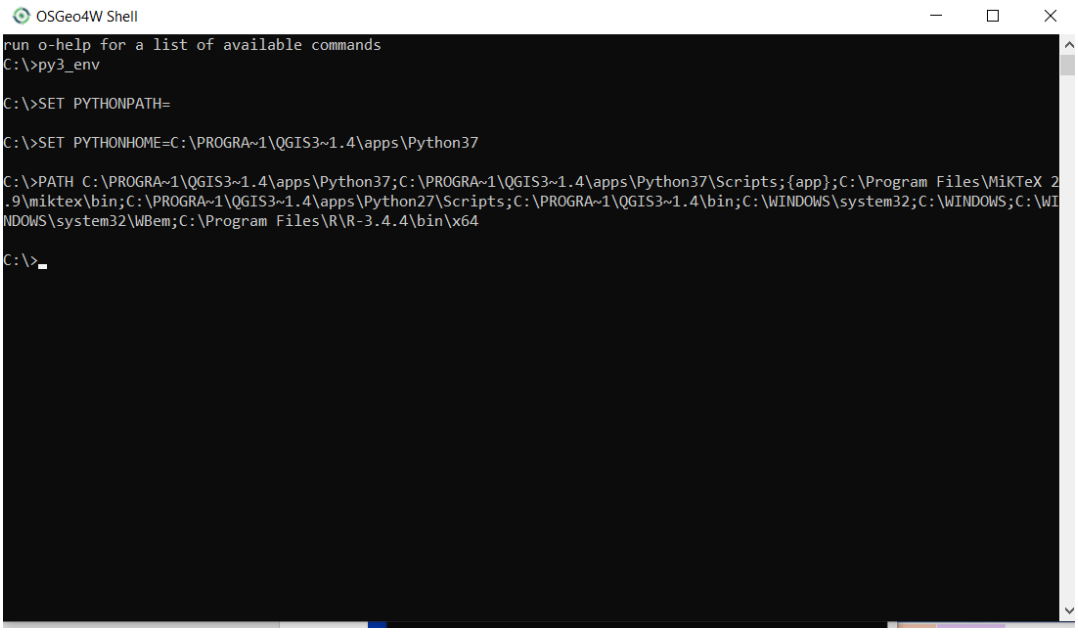
L'interface dessinée ci-dessus n'est pas facile à réaliser, car il faut écrire chaque ligne de codes, connaitre toutes les options possibles si on veut encore plus la personnaliser et surtout mettre les bons paramètres de géométrie pour que tout soit bien aligné. Pour dessiner plus simplement une interface, on peut utiliser Qt Designer qui est installé avec QGIS. En ouvrant le designer, on peut choisir des « templates » plus ou moins complets qui dessinent une partie de l'interface : ici, par exemple, une interface avec des boutons « Ok » et « Annuler » en cliquant sur « Créer ».



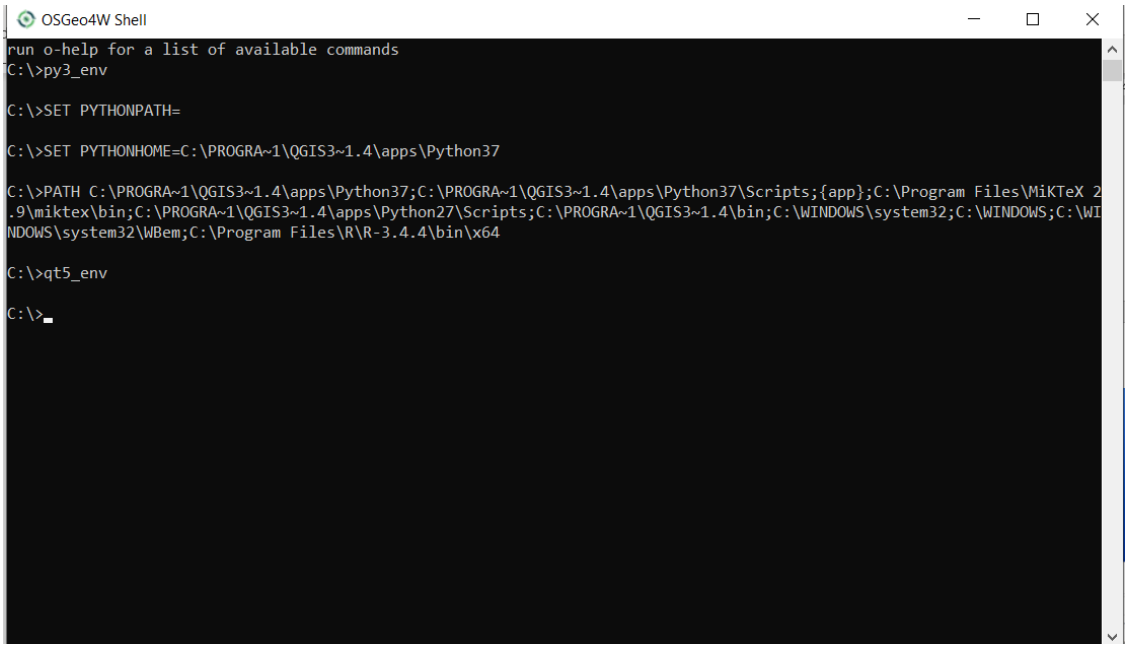
A l'aide de « cliquer-glisser », on peut placer des labels, des comboBox ou encore des lineEdit… A droite, on pourra renommer les objets : par exemple, l'interface en « Dialog_intersection ». On pourra aussi modifier simplement la police et les valeurs numériques.



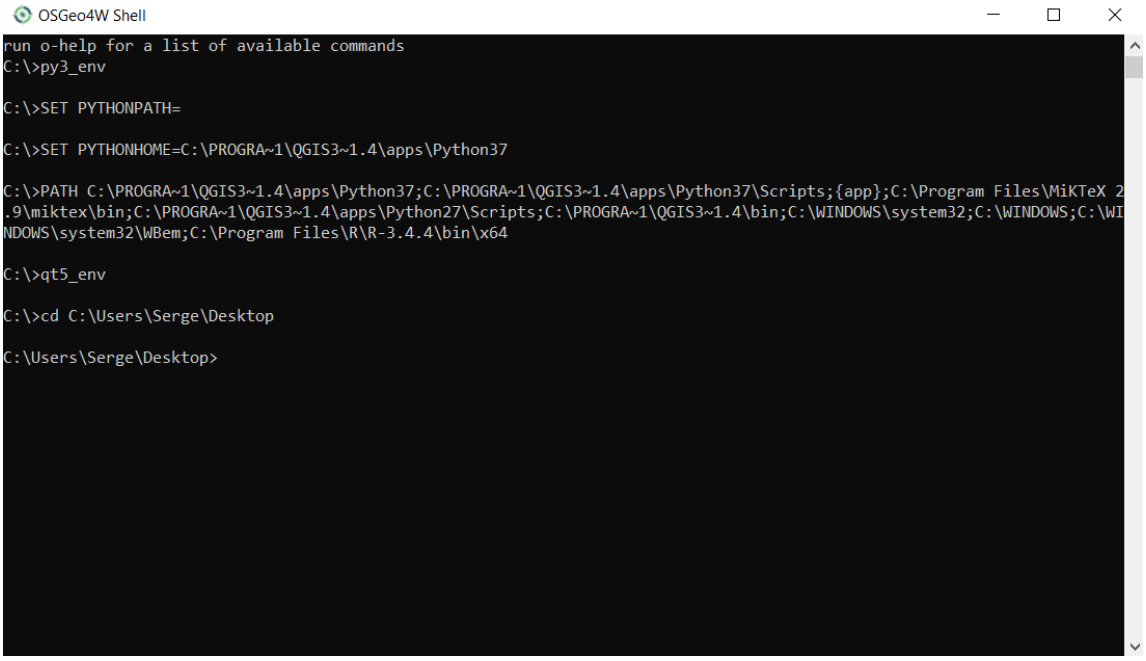
Une fois l'interface dessinée, on peut sauvegarder le résultat. Néanmoins, le résultat est un fichier .ui et non un fichier .py. Il faudra donc le transformer pour l'utiliser en python. On peut en revanche récupérer des fichiers .ui existants pour dessiner plus simplement des interfaces complexes. Pour transformer votre fichier en .py (ici nommé Interface_intersection.ui), il faut utiliser le shell OSGEO. Premièrement, on définit l'environnement python3.



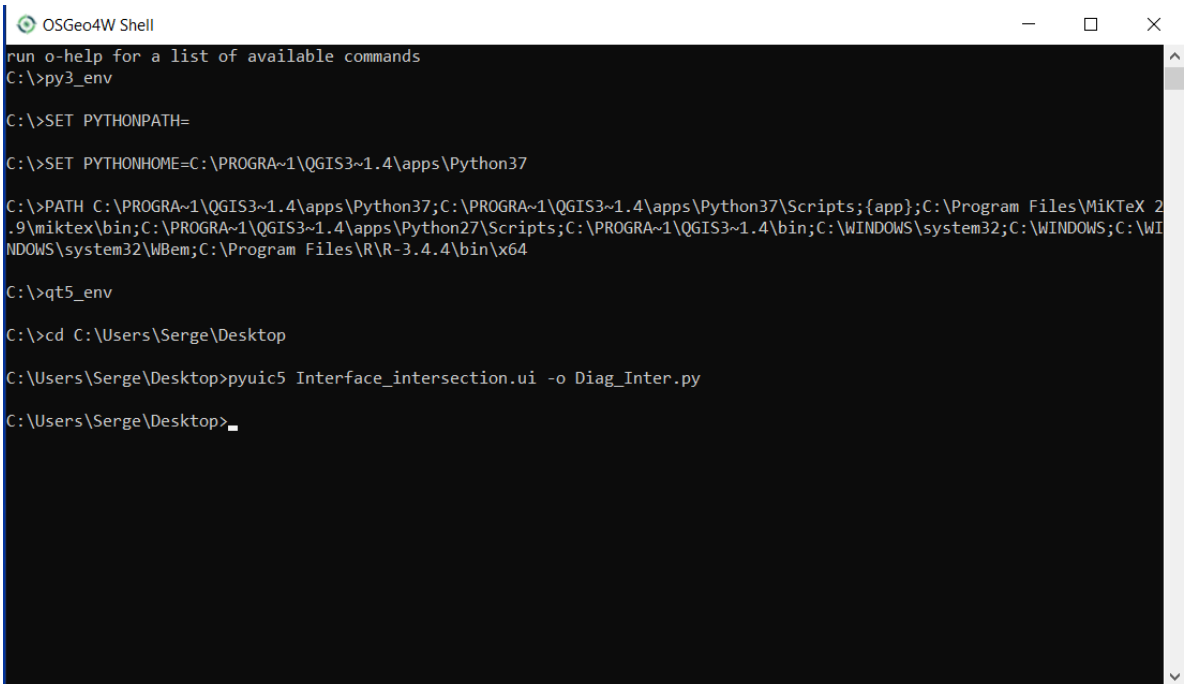
De même, on précise que l'on fait la conversion PyQt5.



Avec la commande « cd », on précise où se trouve notre fichier ui. Ici sur mon bureau.



Enfin on rentre la commande qui transforme le fichier.



Pour utiliser le code python produit, il existe plusieurs solutions. 1) On exécute le fichier python obtenu dans QGIS directement, cela permet de charger les classes créées avant d'exécuter notre code qui pourra s'appuyer sur ces fonctions. 2) On peut copier notre code dans ce nouveau code et trouver comment faire interagir tout ça. 3) On se fonde sur ce code pour l'implémenter dans notre code et redessiner l'interface avec un code plus simple (sans fonction), mais qui en reprend une grande partie.

Si on retient la solution 1), on définit (initialise) l'interface dans notre code à partir de la fonction ui, ici nommée « Ui_Dialog_intersection ».

```
class dialog(QtWidgets.QDialog):
    def __init__(self,parent):
        QtWidgets.QDialog.__init__(self,parent)
        self.ui = Ui_Dialog_intersection()
        self.ui.setupUi(self)
```

On ajoute alors l'interface, puis on l'affiche comme suit.

```
dlg = dialog(iface.mainWindow())
dlg.show()
```

Ensuite, le tour est joué. Il n'y a plus qu'à conserver les éléments que l'on veut modifier dans l'interface (par rapport au code produit pour les boites de dialogue) et supprimer le reste qui est inclut dans le fichier python créé et chargé pour l'interface.

```
dlg = dialog(iface.mainWindow())
dlg.ui.buttonBox.accepted.connect(appui_ok)
dlg.ui.buttonBox.rejected.connect(appui_no)
dlg.ui.comboBox.addItems(nomCouches)
dlg.ui.comboBox_2.addItems(nomCouches)
dlg.show()
```

Et pour que le code se passe bien quand on clique sur « OK », il faut bien récupérer le texte dans l'algorithme.

```
nomZone = dlg.ui.comboBox.currentText()
nomTampon = dlg.ui.comboBox_2.currentText()
```

Ne pas oublier les fermetures des boites de dialogue, on obtient le code suivant. En orange foncé, les grosses modifications effectuées par rapport au code avec l'interface écrite à la main.

```
from PyQt5 import QtCore, QtGui, QtWidgets

def appui_ok() :
    nomZone = dlg.ui.comboBox.currentText()
    nomTampon = dlg.ui.comboBox_2.currentText()
    param = {'INPUT': nomZone,
         'OVERLAY': nomTampon,
         'OUTPUT':  'memory:'
         }
    intersection = processing.run('native:intersection', param)
    QgsProject.instance().addMapLayer(intersection['OUTPUT'])
    intersection['OUTPUT'].setName('Intersection')
    dlg.close()

def appui_no() :
    dlg.close()

couches = qgis.utils.iface.mapCanvas().layers()
nomCouches=[]
for i in range(len(couches)) :
      nomCouches = nomCouches + [couches[i].name()]

class dialog(QtWidgets.QDialog):
    def __init__(self,parent):
        QtWidgets.QDialog.__init__(self,parent)
        self.ui = Ui_Dialog_intersection()
        self.ui.setupUi(self)

dlg = dialog(iface.mainWindow())
dlg.ui.buttonBox.accepted.connect(appui_ok)
dlg.ui.buttonBox.rejected.connect(appui_no)
dlg.ui.comboBox.addItems(nomCouches)
dlg.ui.comboBox_2.addItems(nomCouches)
dlg.show()
```