HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Département Environnement construit & Géoinformation
Filière de Géomatique

HES été

Introduction à la création de plugins QGIS

Sarah Composto

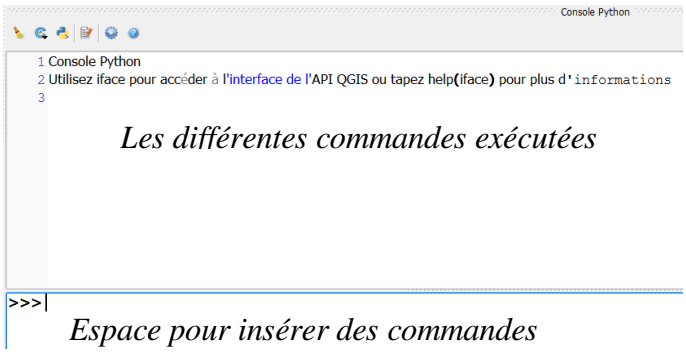
TABLE DES MATIÈRES

1 INTRODUCTION À PYTHON DANS QGIS

QGIS utilise le langage de programmation Python. De plus, dans le cadre de la création de nouveaux plugins, l'API de QGIS est très souvent appelée. C'est grâce à l'API qu'il est possible d'interagir avec les objets (les couches, les objets et l'interface de QGIS). Ce chapitre permet donc de se familiariser avec cette API à travers la console Python de QGIS. D'ailleurs, par l'intermédiaire de cette console, il est possible d'enregistrer les scripts Python afin de pouvoir les réutiliser ultérieurement (par exemple dans un plugin).

1.1 Présentation de la console Python

Aperçu de la console Python



Les différentes commandes exécutées

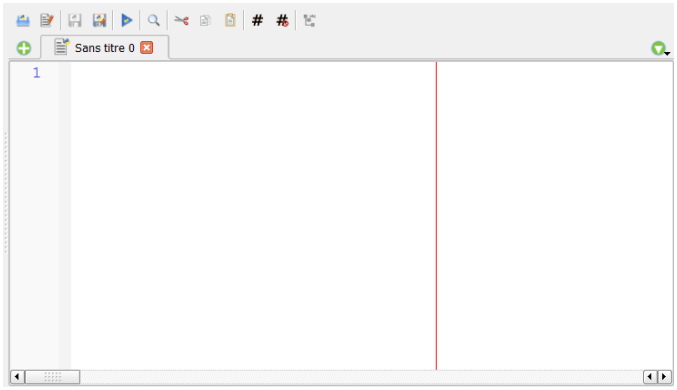
Espace pour insérer des commandes

La console Python présente plusieurs boutons :

- pour effacer la console
- pour importer des classes
- pour exécuter une commande
- pour ouvrir l'éditeur
- pour paramétrer
- pour obtenir de l'aide

Remarque : Effacer la console ne fait qu'effacer l'affichage des commandes exécutées. Ceci signifie que les variables sont conservées.

Aperçu de l'éditeur de la console



L'éditeur de la console présente plusieurs boutons dont notamment :

- pour ouvrir un script Python
- pour enregistrer le script
- pour exécuter le script
- pour commenter du code
- pour décommenter du code

….

1.2 Exercice avec la console Python

1.2.1 Contexte

Cet exercice permet de lire un fichier shapefile, de récupérer non seulement les attributs, mais également la géométrie (latitude et longitude), d'afficher ces différentes caractéristiques et de les écrire dans un fichier TXT. De plus, il permet d'écrire un nouveau fichier shapefile en projetant les données dans un autre système de coordonnées.

1.2.2 Données

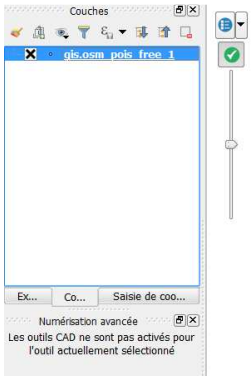
Les données utilisées dans le cadre de cet exercice sont les points d'intérêt d'OpenStreetMap (http://download.geofabrik.de/) au format shapefile.

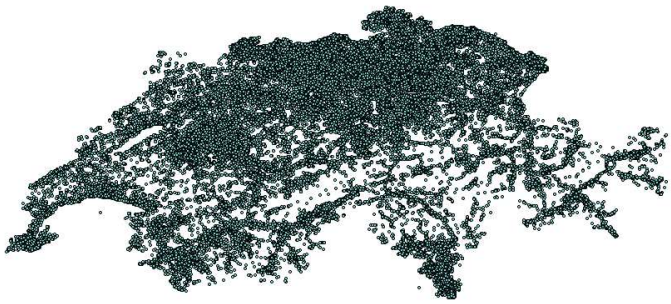
1.2.3 Marche à suivre

1. Préparations préalables :
   - Créer un nouveau dossier pour ce cours sur le disque D: de l'ordinateur avec votre nom : « temp_votreNom »
   - Télécharger les données suisses d'OpenStreetMap au format shapefile à partir du lien suivant :

     http://download.geofabrik.de/

   - Enregistrer les données d'OpenStreetMap dans votre dossier
   - Ouvrir QGIS
   - Créer un nouveau projet QGIS
   - Ajouter la couche des points d'intérêt (« gis.osm_pois_free_1 ») dans ce nouveau projet à partir du menu « Couche » → « Ajouter une couche » :





   - Faire une petite sélection de ces données dans la ville d'Yverdon-les-Bains (échantillon de maximum 300 points)

- Enregistrer la sélection au format shapefile sous un nouveau nom (exemple : « poi_selection »)
- Supprimer la couche de tous les points d'intérêt
- Ajouter la couche précédemment créée (échantillon des points d'intérêt)

2. Ouvrir la console Python de QGIS : menu « Extension » → « Console Python »

Dès à présent, les différentes étapes seront effectuées uniquement par l'intermédiaire de la **console Python** :

3. Récupérer la couche active de ce projet QGIS (ndlr : la couche échantillon des points d'intérêt) grâce à la fonction activeLayer() :

```
layer = iface.activeLayer()
```

Remarque :

- Pour exécuter cette commande depuis la console Python : utiliser la touche « Enter » du clavier.
- « iface » est un objet de la classe QgsInterface qui permet d'accéder au canevas de carte, aux menus, aux barres d'outils, etc.
- « layer » est le nom de la variable dans lequel le résultat de la fonction activeLayer() est stocké. Il est donc possible de nommer cette variable autrement.
- « layer » renvoie à un objet de la classe « QgsVectorLayer » :

```
<qgis._core.QgsVectorLayer object at 0x000000001BF57D90>
```

4. Récupérer les différents éléments (features), c'est-à-dire les différents points d'intérêt de la couche en utilisant la fonction getFeatures() :

```
layer.getFeatures()
```

Astuce : Pour afficher les différentes variables et fonctions possibles pour un objet, il est possible d'utiliser la fonction dir(). Exemple :

```
dir(layer)
```

⇨ La fonction getFeatures() renvoie un objet de type itérateur :

```
<qgis._core.QgsFeatureIterator object at 0x000000001BF57E18>
```

Pour pouvoir récupérer les différents éléments de la couche, il est donc nécessaire d'effectuer une itération sur cet objet, c'est-à-dire une boucle.

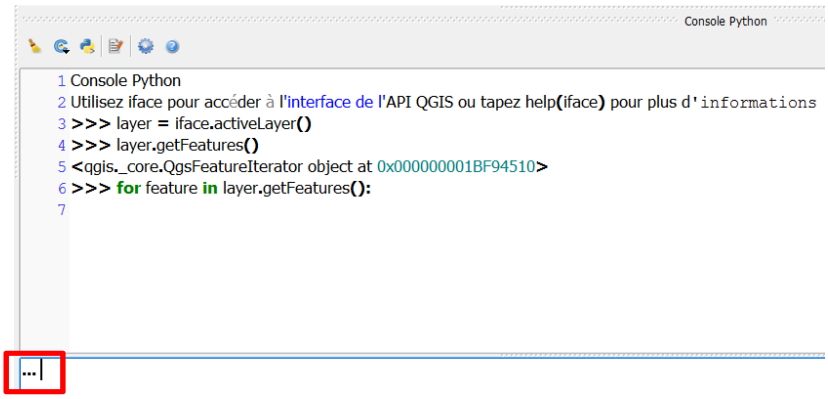
5. Afficher les différents éléments de cette couche en ajoutant une boucle sur la fonction getFeatures() :

```
for feature in layer.getFeatures():
    print feature
```

Attention : Il est nécessaire d'indenter la deuxième ligne afin que Python sache que cette ligne se trouve dans la boucle. Dans la console Python, l'indentation s'effectue avec deux espaces.

Remarque :

- Lorsque la console présente « … » au lieu de « >>> », cela signifie qu'elle est en attente de code (exemple : en attente de la fermeture d'une boucle) :



- Pour clore la boucle : appuyer sur la touche « Enter »

6. A partir du résultat précédent : afficher uniquement le nom de chaque élément (colonne « name ») :

```
for feature in layer.getFeatures():
    print feature['name']
```

⇨ Résultat : La liste des noms s'affiche dans la console.

Remarque :

- « feature['name'] » peut être inséré dans une nouvelle variable (exemple : name).
- Selon le nombre d'éléments qui a été précédemment sélectionné, l'affichage peut prendre plus ou moins de temps.

7. Récupérer la géométrie de chaque élément grâce à geometry() et afficher les géométries sous forme de paires de coordonnées grâce à l'utilisation de la fonction asPoint() :

```
for feature in layer.getFeatures():
    geom = feature.geometry()
    print geom.asPoint()
```

Remarque :

- Pour récupérer uniquement la coordonnée x ou y : utiliser la fonction x() ou y() après la fonction asPoint().
- Il est possible d'ajouter du texte dans l'affichage : `print 'The layer object: ', layer`

  Résultat : `The layer object: <qgis._core.QgsVectorLayer object at 0x000000001BDEE400>`

  Attention : Les différents éléments sont séparés par des virgules.

8. Effectuer un peu de mise en page dans la fonction print afin d'obtenir un résultat semblable à celui-ci :

```
C&A with coord X = 6.6400168 and coord Y = 46.7792502
CFF with coord X = 6.6408634 and coord Y = 46.7814805
Vierino Lauria with coord X = 6.6402715 and coord Y = 46.779728
Shop Express with coord X = 6.641174 and coord Y = 46.7814
NULL with coord X = 6.6405565 and coord Y = 46.7796104
```

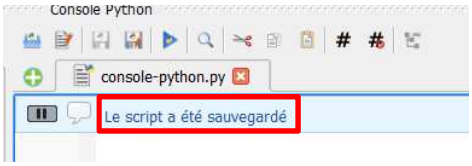
Remarque : La console Python est bien pour tester quelques bouts de code, mais cela devient difficile lorsqu'il y a du code plus compliqué (exemple : présence de boucles). C'est pourquoi, l'éditeur de la console Python est utilisé pour la suite de cet exercice.

9. Ouvrir l'éditeur de la console Python à l'aide du bouton correspondant depuis la console Python

10. Enregistrer un nouveau fichier Python sur votre disque D: à partir du bouton correspondant qui se trouve dans l'éditeur de la console

Remarque :

- Avant de pouvoir enregistrer le fichier Python, QGIS demande d'enregistrer le projet QGIS en lui-même. Si le projet n'avait pas déjà été enregistré, il est donc nécessaire d'effectuer deux enregistrements simultanés. Un message confirme la réussite de l'enregistrement du fichier Python :



- Le fichier Python possède l'extension « .py ».
- Dans le script Python, il est possible d'ajouter des commentaires avec le symbole « # » en début de ligne. A cet effet, le symbole peut être ajouté manuellement (avec la touche correspondante du clavier) ou avec le bouton suivant :

11. Copier le code précédemment créé

    Attention : Ne pas oublier de régulièrement enregistrer le script Python !

    Remarque : Lorsqu'une boucle ou une condition par exemple est créée, l'indentation est automatiquement effectuée par l'éditeur. Si besoin, une tabulation peut être utilisée pour effectuer l'indentation dans l'éditeur de la console.

12. Exécuter le script à l'aide du bouton correspondant et vérifier le résultat dans la console Python

    Remarque : Il est possible qu'il y ait des problèmes d'encodage notamment dans le nom des points d'intérêt. Il s'agit d'un problème lors de l'import de la couche dans QGIS et pour y remédier, il faut se rendre dans les propriétés de la couche (rappel : clic droit sur la couche dans la fenêtre « Couches ») → onglet « Général » → Section « Infos » → Spécifier l'encodage « UTF-8 » :

13. Toujours depuis le script Python, c'est-à-dire depuis l'éditeur de la console Python : ajouter une condition dans la boucle afin de ne pas afficher les données qui ont un nom NULL (astuce : utiliser simplement « if name » pour vérifier qu'il y ait un nom). Le résultat devrait être semblable à celui-ci :

    ```
    C&A with coord X = 6.6400168 and coord Y = 46.7792502
    CFF with coord X = 6.6408634 and coord Y = 46.7814805
    Vierino Lauria with coord X = 6.6402715 and coord Y = 46.779728
    Shop Express with coord X = 6.641174 and coord Y = 46.7814
    ```

14. Écrire le nom et les coordonnées de chaque élément dans un fichier TXT

    Pour ce faire, il est nécessaire de tout d'abord écrire un nouveau fichier en indiquant le chemin et le nom du fichier, ainsi que « w » pour spécifier que le fichier est ouvert en écriture (w pour write) :

    ```
    output = open('C:/Users/sarah.composto/Desktop/codes/poi.txt', 'w')
    ```

    Remarque : Si le fichier n'existe pas, il sera automatiquement créé au lancement de cette commande.

    Attention : Il est également nécessaire de fermer le fichier à la fin du code :

    ```
    output.close()
    ```

    Ensuite, il faut indiquer les éléments à écrire dans le fichier à l'aide de la commande suivante :

    ```
    output.write(line_encode)
    ```

Attention : Il est nécessaire de préalablement indiquer l'encodage utf-8 grâce au code suivant :

```
line_encode = line.encode('utf-8')
```

où « line » est le nom de la variable qui contient les données :

```
line = '%s\n' % (data)
```

Remarque :

- « %s » permet d'indiquer que « data » est une chaîne de caractères (s pour string). Ainsi, pour définir la variable « data », vous pouvez par exemple concaténer différentes chaînes de caractères (string) à l'aide du symbole « + ». Pour transformer des entiers ou des nombres à virgules en chaînes de caractères : utiliser la fonction str().
  Exemples :

  `str(7)` → Pour transformer un entier en chaîne de caractères ; Rappel : Les chaînes de caractères sont entourées de guillemets.

  `data = 'Concaténation' + '!'` → Pour concaténer plusieurs chaînes de caractères et inscrire le résultat de cette concaténation dans une nouvelle variable appelée, dans cet exemple-ci, « data »

- « \n » permet d'ajouter un retour à la ligne ; Ainsi, chaque élément est alors écrit sur une nouvelle ligne dans le fichier texte.

⇨ Aperçu du résultat :

```
C&A with coord X = 6.6400168 and coord Y = 46.7792502
CFF with coord X = 6.6408634 and coord Y = 46.7814805
Vierino Lauria with coord X = 6.6402715 and coord Y = 46.779728
Shop Express with coord X = 6.641174 and coord Y = 46.7814
Bâtiment l'Etoile with coord X = 6.6391313 and coord Y = 46.7785005
C&A with coord X = 6.6404372 and coord Y = 46.7796595
```

15. Enregistrer les points d'intérêt au format shapefile et en les projetant dans le système suisse EPSG:21781 :

Pour définir un système de projection :

```
dest_crs = QgsCoordinateReferenceSystem(21781, QgsCoordinateReferenceSystem.EpsgCrsId)
```

Pour enregistrer les données au format shapefile et en les projetant dans le système de projection préalablement indiqué :

```
QgsVectorFileWriter.writeAsVectorFormat(layer, 'C:/Users/sarah.composto/Desktop/codes/poi.shp', 'utf-8', dest_crs, 'ESRI Shapefile')
```

Remarque : Il est possible de récupérer le système de coordonnées de la couche avec la commande suivante :

```
layer.crs().authid()
```

crs() retourne un objet de QgsCoordinateReferenceSystem. Avec authid(), le code EPSG est retourné.

16. Vérifier le fichier shapefile et, le cas échéant, ne pas oublier d'enregistrer le script Python

Remarque : L'éditeur de la console Python permet d'écrire de grands scripts Python réutilisables. Toutefois, il ne permet pas la création d'interfaces utilisateurs, permettant ainsi des échanges plus simples avec les utilisateurs. Les prochains chapitres se consacrent à la création de plugins, des scripts Python accompagnés d'interfaces graphiques.

2 CRÉATION D'UN PLUGIN QGIS

2.1 Introduction

Il y a deux types principaux de plugins :

- Les plugins qui sont développés par l'équipe de développement de QGIS et qui sont automatiquement intégrés dans chaque distribution de QGIS.
- Les plugins dits « externes » qui ne sont pas automatiquement intégrés dans les distributions de QGIS, mais qui peuvent être téléchargés depuis le répertoire officiel de QGIS (http://plugins.qgis.org/plugins/). Une fois téléchargés, les plugins externes se trouvent dans le dossier : C:/Utilisateurs/xxx/.qgis2/python/plugins où xxx est votre nom d'utilisateur. Il est possible d'installer / de désinstaller / de mettre à jour des plugins depuis le menu « Extension » de QGIS → « Installer / Gérer les extensions ».

La création d'un plugin QGIS peut être divisée en deux parties. La première partie concerne l'interface graphique du plugin, tandis que la deuxième partie concerne l'implémentation concrète du plugin.

- Pour créer l'interface graphique du plugin, il existe plusieurs solutions. Dans le cadre de cet exercice, le logiciel QT Designer est utilisé. Ce logiciel permet de dessiner graphiquement les différents éléments, puis de pouvoir générer de manière automatique le code correspondant à ces éléments.

- Le plugin est constitué de différents dossiers / fichiers avec une structure particulière :

__init__.py	Fichier Python qui permet d'importer les packages Python
metadata.txt	Fichier texte composé des métadonnées (→ informations) sur le plugin : nom, description, etc. du plugin
xxx.py	Où xxx est le nom de votre plugin Fichier Python → code principal du plugin → C'est dans ce fichier que se trouvent les différentes fonctions du plugin : __init__() pour accéder à l'interface QGIS, initGui() est appelé lorsque le plugin est chargé, unload() est appelé lorsque le plugin est déchargé, ….
xxx.ui	Fichier qui contient l'interface graphique du plugin
…	…

Pour plus d'informations :
http://docs.qgis.org/2.18/en/docs/pyqgis_developer_cookbook/plugins.html#developing-plugins

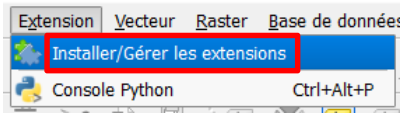
Rappel : Les plugins installés se trouvent dans le dossier :
C:/Utilisateurs/xxx/.qgis2/python/plugins où xxx est votre nom d'utilisateur.

2.2 Installations

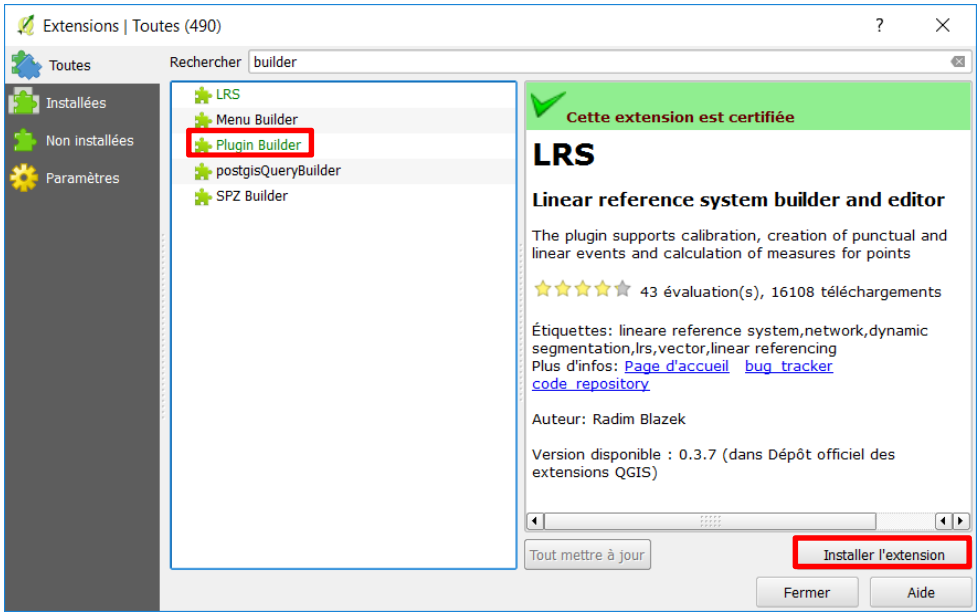
Afin de créer de nouveaux plugins QGIS, il est nécessaire de posséder les éléments suivants sur son ordinateur :

- QGIS : http://qgis.org qui comprend notamment QT Designer, permettant ainsi la création simplifiée d'interfaces graphiques pour les plugins
- L'extension « Plugin Builder » (https://plugins.qgis.org/plugins/pluginbuilder/) qui permet la création du squelette de plugins (dossiers, fichiers)
- L'extension « Plugin Reloader » (https://plugins.qgis.org/plugins/plugin_reloader/) qui permet de recharger les modifications du plugin sans avoir besoin de redémarrer QGIS

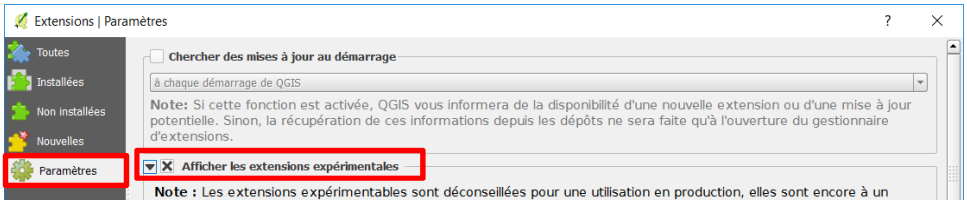
Dans le cadre de ce cours, il est nécessaire d'ajouter les plugins susmentionnés dans QGIS. Pour ce faire :



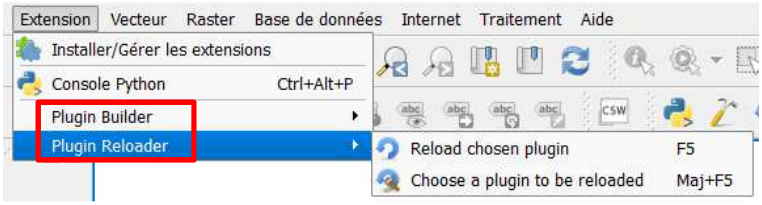
A partir de la fenêtre « Extensions » : chercher les plugins susmentionnés dans la barre de recherche et les installer :



Attention : Pour le plugin Reloader, il est nécessaire de cocher « Afficher les extensions expérimentales » dans l'onglet « Paramètres » de la fenêtre « Extensions » :



⇨ Résultat : Ces deux plugins ont chacun ajouté une nouvelle entrée dans le menu « Extension » de QGIS :



De plus, plusieurs boutons se sont également ajoutés dans l'interface de QGIS :

Plugin Builder

Plugin Reloader

2.3 Exercice de création d'un plugin

2.3.1 Contexte

Cet exercice présente les différentes étapes de création d'un plugin QGIS. Le plugin QGIS créé dans le cadre de cet exercice permet de sélectionner, par l'intermédiaire d'une interface graphique, une des couches ouvertes dans le projet QGIS et d'afficher quelques informations relatives à cette couche sélectionnée (exemples : type de couche, type de géométrie si la couche est de type vectorielle, système de coordonnées, unité du système de coordonnées).

2.3.2 Données

N'importe quels jeux de données peuvent être utilisés dans le cadre de cet exercice. Toutefois, afin de pouvoir tester le plugin, il est préférable d'avoir plusieurs jeux de données de différents types (raster, vecteur – points, vecteur – polylignes, vecteur – polygones).

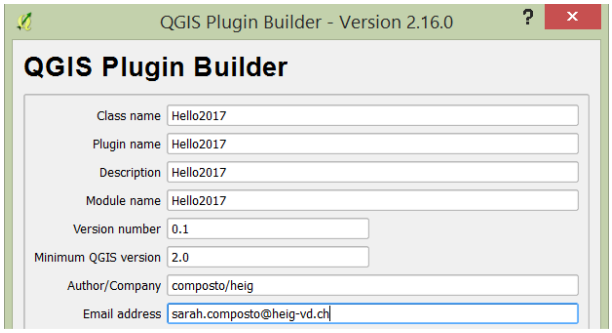
Il vous est ainsi possible de, par exemple, utiliser les données d'OpenStreetMap :

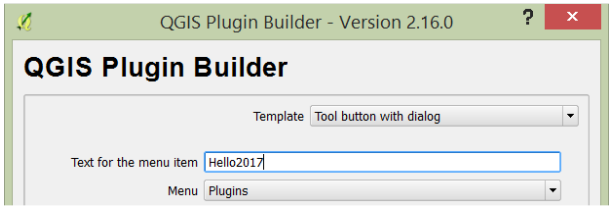
http://download.geofabrik.de/

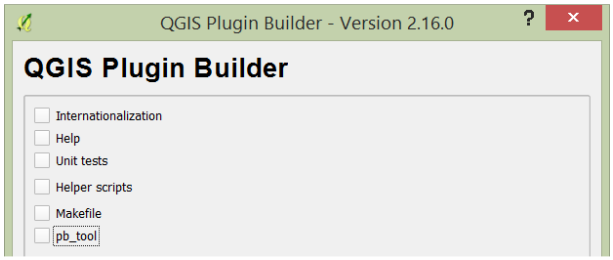
2.3.3 Marche à suivre – Structure du plugin

Dans un premier temps, il s'agit de créer la structure des dossiers / fichiers du plugin à l'aide de « Plugin Builder ». Ensuite, l'interface graphique est modifiée et le code Python correspond est inséré.

1. Créer la structure des dossiers / fichiers à partir du menu « Extension » → « Plugin Builder » → « Plugin Builder » :







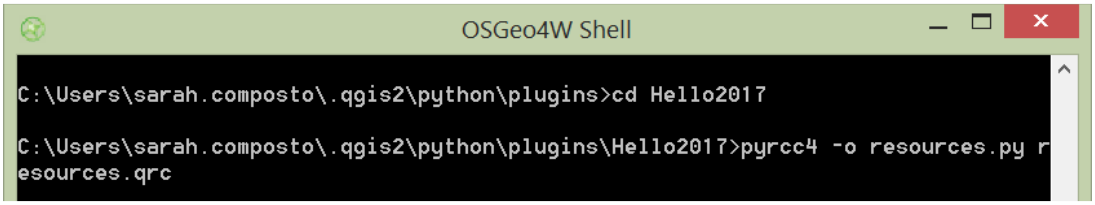
   Sélectionner le dossier C:/Utilisateurs/xxx/.qgis2/python/plugins où xxx est votre nom d'utilisateur.

⇨ Résultat : La structure des dossiers / fichiers du plugin a été créée.

2. Par l'intermédiaire d'une invite de commandes Windows (cmd) : se rendre dans le dossier du plugin et y générer le fichier Python avec les ressources grâce à la commande suivante :

```
C:\Users\sarah.composto\.qgis2\python\plugins\Hello2017>pyrcc4 -o resources.py resources.qrc
```

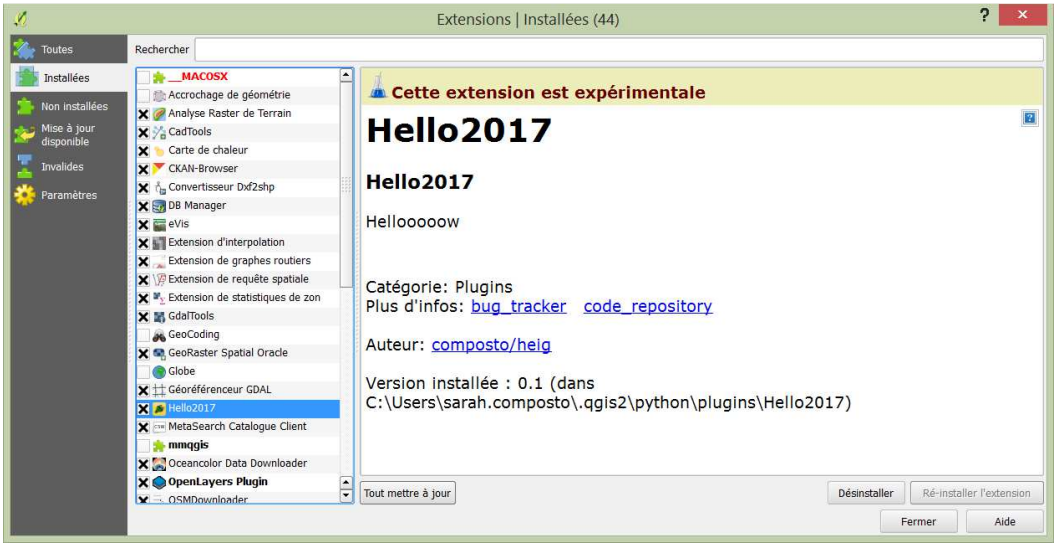
Remarque : Il est également possible de compiler le fichier « resources.qrc » à l'aide d'OSGeo4W Shell, un fichier BAT du logiciel QGIS. Pour cela : lancer OSGeo4W Shell, se rendre dans le dossier du plugin à l'aide de commandes « cd … » et exécuter la commande :



⇨ Résultat : Un nouveau fichier (« resources.py ») s'est créé dans le dossier du plugin. Celui-ci contient la traduction en Python du fichier « resources.qrc », un document xml qui contient les liens vers les ressources extérieures (exemple : icône).

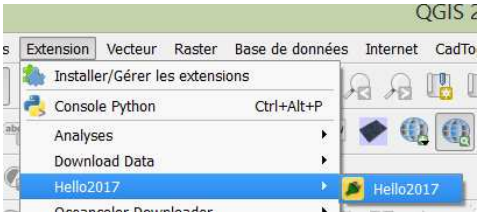
3. Redémarrer QGIS

Remarque : Le plugin se trouve dans le menu « Extension » → « Installer / Gérer les installations » → Onglet « Installées » :



4. Dans le menu « Extension » de QGIS : ouvrir le plugin (ou avec le bouton) :

Entrée de menu Bouton



⇨ Résultat : Pour l'instant, l'interface graphique du plugin n'est composée que de deux boutons :

2.3.4 Marche à suivre – Interface graphique du plugin

Il s'agit à présent de modifier l'interface graphique grâce au logiciel « Qt Designer with QGIS ». Attention : Afin de voir les modifications effectuées sur l'interface graphique, il est nécessaire de recharger le plugin. Ceci peut se faire en redémarrant QGIS ou en utilisant le « Plugin Reloader ».

1. Par l'intermédiaire d'un explorateur Windows : se rendre dans le dossier du plugin : C:/Utilisateurs/xxx/.qgis2/python/plugins où xxx est votre nom d'utilisateur

2. Ouvrir le fichier ayant l'extension « .ui » avec le logiciel « Qt Designer with QGIS » : clic droit sur le fichier → Propriétés → Onglet « Général » → Bouton « Modifier » :

Sélectionner « designer.exe » dans C:\Program Files\QGIS_Nodebo\bin. Une fois les nouvelles propriétés du fichier sauvegardées : double-cliquer sur le fichier pour l'ouvrir avec « Qt Designer with QGIS ».

Remarque : L'interface graphique se trouve dans un fichier avec l'extension « .ui » qui se présente sous la forme d'un fichier XML. Les différentes balises qu'il contient correspondent aux différents éléments de l'interface graphique. Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Hello2017DialogBase</class>
 <widget class="QDialog" name="Hello2017DialogBase">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>400</width>
    <height>300</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Hello2017</string>
  </property>
  <widget class="QDialogButtonBox" name="button_box">
   <property name="geometry">
    <rect>
     <x>30</x>
     <y>240</y>
     <width>341</width>
     <height>32</height>
```

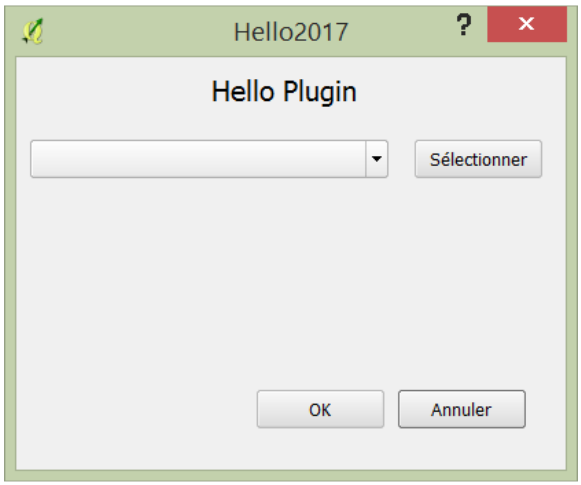
Remarque : Chaque bouton, chaque étiquette, chaque liste, etc. est un widget dans le fichier XML.

3. A l'aide des fenêtres « Boîte de widget » et « Editeur de propriétés » : ajouter de nouveaux éléments dans l'interface graphique du plugin (voir ci-dessous) et en parallèle, visualiser le résultat sous QGIS.

   Remarque :
   - Pour ouvrir les fenêtres « Boîte de widget » et « Editeur de propriétés » dans Qt Designer : utiliser le menu « Affichage » du logiciel.
   - Pour visualiser les modifications de l'interface graphique dans QGIS : utiliser le « Plugin Reloader » dans le menu « Extension » de QGIS → « Plugin Reloader » ou avec le bouton correspondant :

   Attention : Ne pas oublier de régulièrement enregistrer les modifications de l'interface graphique (menu « Fichier » → « Enregistrer »).

Les éléments à ajouter dans l'interface graphique :



Un titre avec « Label » dans la section « Display Widgets » → Le titre est à centrer et à mettre en police 12.

Une liste déroulante avec « Combo Box » dans la section « Input Widgets »

Un bouton avec « Push Button » dans la section « Buttons » qui s'appelle « Sélectionner »

Remarque : La liste déroulante est pour l'instant vide et sera, dans le cadre du prochain chapitre, remplie des différentes couches présentes dans le projet QGIS.

2.3.5 Marche à suivre – Code Python du plugin

A présent, l'interface graphique possède plusieurs éléments d'interaction, mais ceux-ci ne sont actuellement pas utilisables. Afin de pouvoir créer l'interaction avec les utilisateurs, il est nécessaire de modifier le code Python qui se trouve dans le fichier Python principal (ici : « Hello2017.py »), plus précisément dans la fonction run() de ce fichier.

Introduction

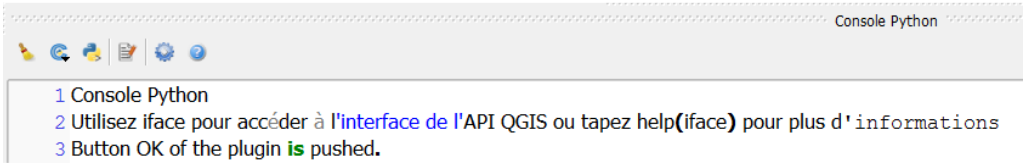
1. Pour vérifier que le plugin réponde bien : ajouter tout d'abord un « print … » dans la condition « if » avec le logiciel Notepad++ par exemple :

```
def run(self):
    """Run method that performs all the real work"""
    # show the dialog
    self.dlg.show()
    # Run the dialog event loop
    result = self.dlg.exec_()
    # See if OK was pressed
    if result:
        # Do something useful here - delete the line containing pass and
        # substitute with your code.
        print 'Button OK of the plugin is pushed.'
        pass
```

Remarque : En Python, les fonctions sont définies avec le mot-clé « def ».

2. Depuis QGIS : recharger le plugin avec « Plugin Reloader », ouvrir la console Python et tester le fonctionnement du plugin

⇨ Résultat : Le texte est inscrit dans la console Python lorsque le bouton « OK » est appuyé :



Remarque : Le mot « is » est différent des autres mots, car celui-ci est un mot-clé de Python. Toutefois, comme il est considéré comme une chaîne de caractères dans ce cas-ci, il peut sans problème être utilisé.

Affichage des couches dans la liste déroulante de l'interface graphique

3. Ajouter une nouvelle fonction (exemple de nom : « loadActiveLayers ») dans le fichier Python principal (ici : « Hello2017.py ») et appeler cette nouvelle fonction depuis la fonction « run » :

```
self.loadActiveLayers()
```

4. Récupérer les couches qui ont été préalablement chargées dans QGIS avec les deux lignes de code suivantes :

```
canvas = iface.mapCanvas()
activeLayers = canvas.layers()
```

Attention : Afin de pouvoir utiliser « iface », il est nécessaire d'ajouter l'import suivant en haut du fichier Python principal :

```
from qgis.utils import *
```

Remarque :

- « iface » est un objet de la classe QgsInterface qui permet d'accéder au canevas de carte, aux menus, aux barres d'outils, etc. tandis que la fonction mapCanvas() retourne un objet de type QgsMapCanvas.
- La deuxième ligne de code permet de récupérer toutes les couches de l'objet « canvas » présentes dans le projet QGIS actuellement ouvert. « activeLayers » est une variable qui contient donc une liste de toutes les couches présentes.

5. Récupérer et afficher le nom de chaque couche à partir de la variable « activeLayers » précédemment définie et par l'intermédiaire d'une boucle :

```
for layer in activeLayers:
```

6. En utilisant la fonction name() dans la boucle précédemment créée, il est possible de récupérer le nom de chaque couche :

```
layerName = layer.name()
```

Remarque : « layer » représente chacune des couches à tour de rôle. Ainsi, au premier tour de la boucle, la variable « layer » contient les données de la première couche. Au deuxième tour, « layer » contient les données de la deuxième couche, etc.

7. Toujours dans la boucle : récupérer à présent le type de fichier de chaque couche avec la fonction type()

Remarque : La fonction type() retourne une valeur de « 0 » pour une couche vectorielle et une valeur de « 1 » pour une couche de type raster.

8. Toujours au sein de la boucle : créer une nouvelle variable (exemple de nom : « type ») et la définir de la manière suivante :

Si la fonction type() retourne 0 => type = « vecteur »

Si la fonction type() retourne 1 => type = « raster »

Astuce : Utiliser une condition (if: … elif: … else: …).

Attention : Il est nécessaire de convertir le résultat de la fonction type() en string avec l'aide de la fonction str().

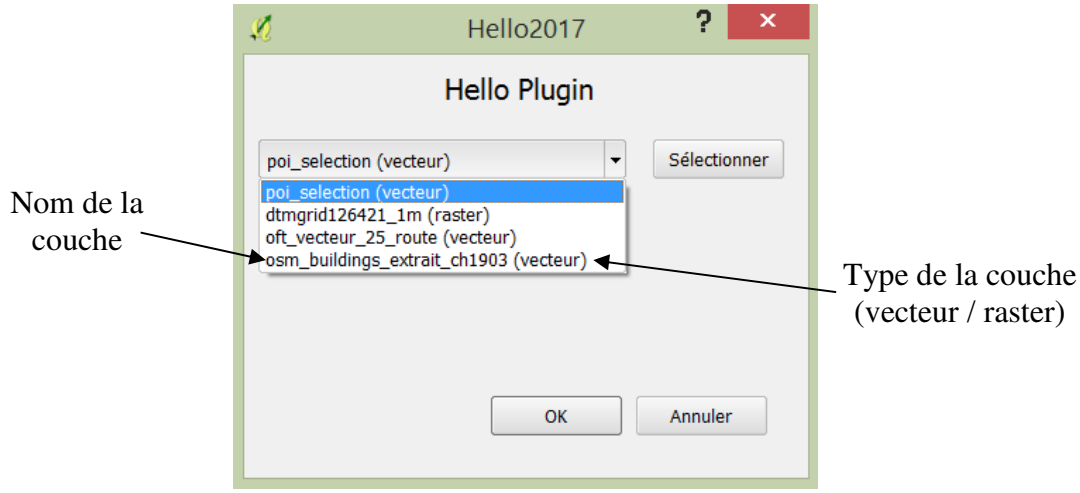
9. Toujours dans la boucle : ajouter la ligne de code suivante afin de pouvoir ajouter le nom et le type de chacune des couches dans la liste déroulante (ici : « comboBox »)

```
self.dlg.comboBox.addItem(layerName)
```

Attention : « comboBox » fait référence au nom de la liste déroulante qui a été définie dans le fichier d'extension .ui :

```
<widget class="QComboBox" name="comboBox">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>60</y>
   <width>261</width>
   <height>28</height>
  </rect>
 </property>
 <property name="font">
```

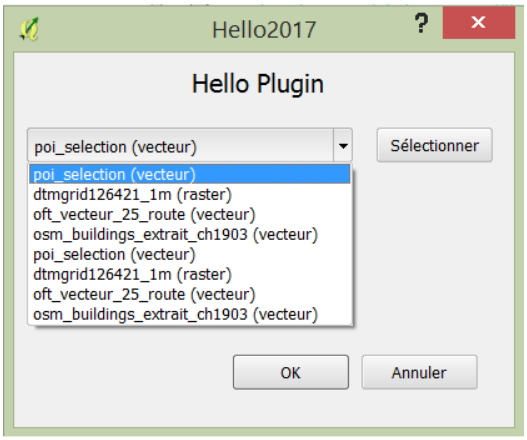
Exemple du résultat attendu :



10. Tester le plugin dans QGIS

Attention : Ne pas oublier de charger les modifications par l'intermédiaire de « Plugin Reloader ».

Problème : En exécutant plusieurs fois la fonction de chargement des couches (ici : « loadActiveLayers »), les différentes couches s'ajoutent à double, à triple, etc. :



Pour y remédier, il suffit d'effectuer un nettoyage de la liste déroulante au début de la fonction (ici : « loadActiveLayers ») :

```
self.dlg.comboBox.clear()
```

Récupération de la couche sélectionnée par l'utilisateur

11. Dans le fichier Python principal : créer une nouvelle fonction (exemple de nom : « makeSomeStatistics ») qui est appelée lors du clic sur le bouton « Sélectionner »

12. Récupérer le clic sur le bouton « Sélectionner » dans la fonction « add_action » du fichier Python principal :

```
self.dlg.pushButton.clicked.connect(self.makeSomeStatistics)
```

Attention : « pushButton » fait référence au nom du bouton « Sélectionner » qui a été défini dans le fichier d'extension .ui :

```
<widget class="QPushButton" name="pushButton">
 <property name="geometry">
  <rect>
   <x>290</x>
   <y>60</y>
   <width>93</width>
   <height>28</height>
  </rect>
 </property>
 <property name="font">
```

« makeSomeStatistics » est le nom de la fonction qui est exécutée lorsque le bouton « pushButton » est cliqué (« clicked »).

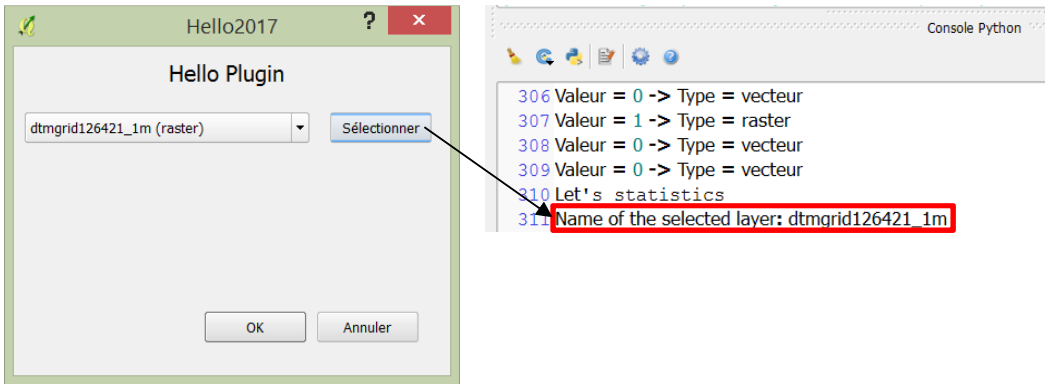
13. Dans la nouvelle fonction précédemment créée : récupérer la couche sélectionnée par l'utilisateur en récupérant l'index de la couche dans la liste déroulante :

```
index = self.dlg.comboBox.currentIndex()
```

Remarque : A partir de cet index, il est possible de récupérer l'objet qui a été sélectionné par l'utilisateur :

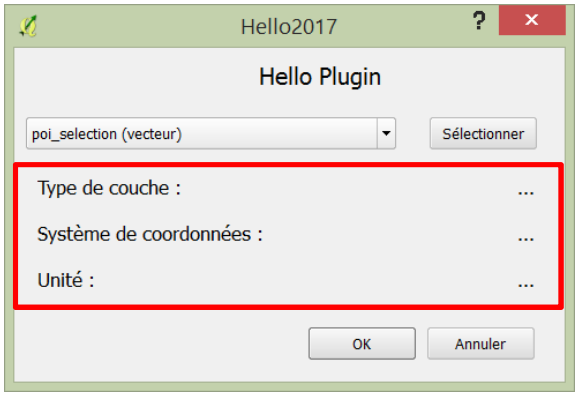
```
canvas = iface.mapCanvas()
selectedLayer = canvas.layers()[index]
print selectedLayer.name()
```

⇨ Exemple du résultat attendu :



Affichage de quelques informations relatives à la couche sélectionnée

14. Modifier l'interface graphique de manière à y afficher quelques informations relatives à la couche sélectionnée :



Dans ce cas-ci, des « Label » (de la section « Display Widgets ») ont été utilisés non seulement pour les textes déjà écrits (exemple : « Type de couche : »), mais également pour les textes, représentés par « … », qui seront remplis par l'intermédiaire de code Python.

Voici quelques idées d'informations qui pourraient être affichées :

- Type de géométrie : ponctuel, linéaire, surfacique
- Système de coordonnées : EPSG 21781 par exemple
- Unité de la couche : mètre par exemple
- Nombre d'objets dans la couche
- Noms des différents champs de la table attributaire
- Etendue géographique de la couche
- …

15. Récupérer les différentes informations (exemples : le type de couche, la géométrie de la couche si celle-ci est de type vectoriel, le code EPSG du système de coordonnées, l'unité du système de coordonnées) à l'aide des fonctions respectives (s'aider d'Internet pour trouver les différentes fonctions) ; Exemple :

```
str(selectedLayer.geometryType())
```

Remarque :

- La fonction str() permet de convertir le résultat qui est de type `<class 'qgis._core.GeometryType'>` en chaîne de caractères.

  Pour connaître le type d'une variable : utiliser la fonction type() ; Exemples :

  `type('Exemple de chaîne de caractères')` renvoie : `<type 'str'>` => type : str, c'est-à-dire chaîne de caractères (string)

  `type(3)` renvoie : `<type 'int'>` => type : int, c'est-à-dire entier (integer)

- La fonction geometryType() permet de connaître le type de géométrie si la couche est vectorielle, c'est-à-dire s'il s'agit de données ponctuelles, linéaires ou surfaciques. Attention : Si la couche n'est pas vectorielle (ndlr : si la couche est de type raster), le code renvoie une erreur. => Dans le code Python, il est donc important de tester (à l'aide de la fonction type() et l'ajout d'une condition) le type de géométrie avant d'appliquer la fonction geometryType() afin de ne pas l'appliquer à une couche de type raster :

```
layerType = str(selectedLayer.type())
if layerType == '1':
  geomTxt = 'Raster'
elif layerType == '0':
  geomTxt = 'Vecteur'
else:
    geomTxt = '?'
```

16. Afficher ces différentes informations dans l'interface graphique en sélectionnant l'élément graphique en question (exemple : le label xxx) :

```
self.dlg.label_5.setText(geomTxt)
```

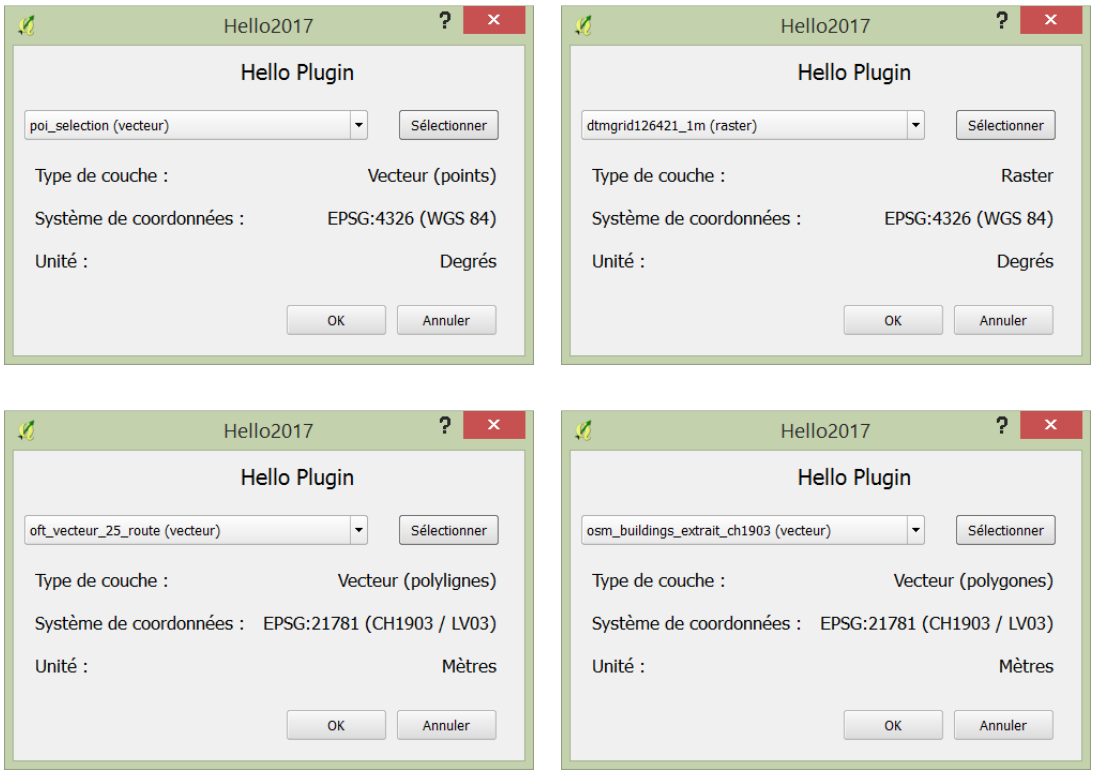
Remarque :

- « label_5 » est le nom du label dans le fichier .ui.
- La fonction setText() permet d'ajouter du texte au label en question. Le paramètre de la fonction setText() (ndlr : la valeur entre les parenthèses après la fonction ; ici : variable qui a été nommée « geomTxt ») est le texte en question.

Attention : Ne pas oublier d'effectuer un nettoyage du texte de chaque label grâce à la fonction clear() qui est à placer au début de la fonction (ici : « makeSomeStatistics ») :

```
self.dlg.label_5.clear()
```

⇨ Exemple du résultat attendu :



2.3.6 Marche à suivre – Personnalisation du plugin

Pour changer l'icône du plugin

1. Télécharger une image (exemple : sur Internet)

   Attention : Choisir une image lisible au format icône !

2. Sauvegarder l'image dans le dossier du plugin

   Attention : L'image doit être de résolution 24 x 24 pixels. Si besoin, il est possible de redimensionner une image à l'aide de Gimp → Menu « Image » → « Echelle et taille de l'image… ».

3. Avec un éditeur de texte (exemple : Notepad++) : modifier le fichier « resources.qrc » pour y ajouter le lien vers l'image :

```
<RCC>
    <qresource prefix="/plugins/Hello2017" >
        <file>cat.png</file>
    </qresource>
</RCC>
```

4. Depuis une invite de commandes Windows : générer le fichier Python avec les ressources grâce à la commande suivante :

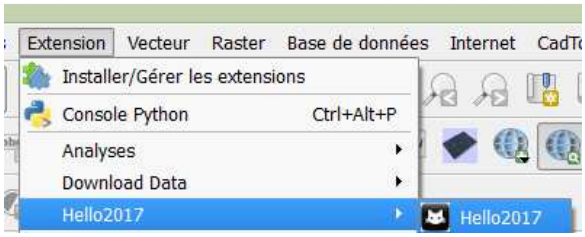
```
C:\Users\sarah.composto\.qgis2\python\plugins\Hello2017>pyrcc4 -o resources.py resources.qrc
```

5. Dans le fichier Python principal du plugin (ici : « Hello2017.py ») → dans la fonction initGui() : changer le nom de l'image avec Notepad++ par exemple :

```
icon_path = ':/plugins/Hello2017/cat.png'
```

6. Dans QGIS : recharger le plugin avec « Plugin Reloader »

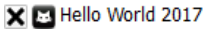
⇨ Résultat : La nouvelle icône apparaît sur le bouton du plugin et dans le menu « Extension » :



Remarque : Pour changer l'icône dans la fenêtre des extensions (menu « Extension » → « Installer / Gérer les extensions ») : il faut modifier le fichier metadata.txt du plugin :

```
homepage=
category=Plugins
icon=cat.png
# experimental flag
experimental=False
```

⇨ Résultat :



Attention : Pour mettre à jour les modifications du fichier metadata.txt, il est nécessaire de redémarrer QGIS.

3 ANNEXES

Quelques liens Internet pour obtenir plus d'informations :

- http://geotribu.net/node/284
- Site officiel de QGIS :
  - http://docs.qgis.org/2.18/en/docs/user_manual/plugins/plugins_index.html
  - http://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/index.html
- Documentation de l'API de QGIS : http://qgis.org/api/2.18/