



Formation

Module IV



Formation Python pour QGIS 3

Didier LECLERC
Conseiller en Management des
Systèmes d'Information Géographique
Département Relation Client

SG / SNUM / UNI / DRC



Cyril HOUISSE
Chef de projet usages et accompagnement
décisionnel, datavisualisation, datascience
Et intelligence artificielle

SG/SNUM/MSP/DS/GD3IA/PUAD



MINISTÈRE
DE LA TRANSITION
ÉCOLOGIQUE ET SOLIDAIRE
www.ecologique-solidaire.gouv.fr

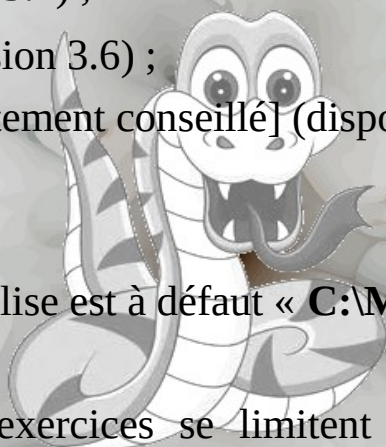
MINISTÈRE
DE LA COHÉSION
DES TERRITOIRES
www.cohesion-territoires.gouv.fr



Pré-requis logiciel :

Pour dérouler correctement cette valise, vous devez au préalable avoir installé :

- **QGIS** [obligatoire] (version 3.x) ;
- **PYTHON** [facultatif] (version 3.6) ;
- **PLUGIN RELOADER** [fortement conseillé] (disponible dans la valise – Kit)



Le répertoire d'installation de la valise est à défaut « **C:\MOC_Q3_STAGE** ».

Les jeux de données pour les exercices se limitent à ceux contenus dans le projet « **test_QGIS3.qgs** » placé dans le répertoire « **Donnees** ».

La présente valise repose sur de multiples sources documentaires disponibles sur internet, et sur l'expérience capitalisée autour de développements d'extensions pour QGIS. La plupart des sources sont indiquées. Merci à l'ensemble des contributeurs volontaires ou involontaires pour le présent support.



Conventions sur la valise :



La présence de ce pictogramme indique qu'un exercice doit être réalisé (cf. le répertoire « **Exercices** » de la valise). « **A faire** » [Obligatoire]



La présence de ce pictogramme indique qu'un exercice sera fait en démonstration par les formateurs



Zone de code
exemple

Les codes exemples

#Exemple

Les mots ou caractères importants sont en gras de couleur magenta.

Module I :

- Python, un peu d'histoire ...
- Python, généralités ...
- Python, les éditeurs ...
- Python, les caractéristiques du langage ...
- Python, premiers contacts avec le langage ...

Module II :

- Python, un langage orienté objet ...
- Python, un langage puissant ...
- Python, gérer les erreurs ...
- Python, de multiples bibliothèques et outils ...
- Python, créer des interfaces graphiques ...
- Python, aller plus loin ...



De Python à QGIS ... (*pont*)

Module III :

- QGIS, un peu d'histoire ...
- QGIS, premiers contacts avec l'API ...
- QGIS, les interfaces pour Python

Module IV :

- QGIS, créer une extension ...
- QGIS, connecter des actions ...
- QGIS, manipuler les objets des couches ...
- QGIS, aller plus loin ...



QGIS, Créer une extension ...

Concepts



QGIS, Créer une extension ...

Même si les possibilités offertes par la console Python sont intéressantes et se révéleront parfois suffisantes pour une automatisation simple, il est parfois indispensable de produire un développement sous forme d'une extension pour QGIS.

La réflexion sur l'écriture d'une extension doit toujours se poser (opportunité, périmètre fonctionnel, organisation des modules ⁽¹⁾, implémentation de classes, maquettage des IHM ...).

Les diapos qui suivent présentent les grands principes structurants pour la réalisation d'une extension. Ces présentations ne se veulent pas le reflet de toutes les possibilités offertes par **Python**, **QGIS** et **Qt**.

⁽¹⁾ Bien organisé son code (structuration, documentation, mutualisation de fonctions génériques ...) est un élément d'urbanisation puisqu'il concourt à une maintenance plus facile du développement.



QGIS, Créer une extension ...

Un certain nombre d'éléments sont indispensables pour implémenter correctement une extension pour **QGIS** :

- Un répertoire dédié à l'extension ;
- Un fichier « *metadata.txt* » (obligatoire et structuré, c'est le descriptif de l'extension : métadonnées) ;
- Un fichier « *__init__.py* » (c'est le lanceur de l'extension) ;
- Un fichier « *main.py* » (c'est le constructeur primaire de l'extension, ici nommé main mais l'usage courant est de donner le nom du répertoire de l'extension) : à défaut, il comprend au moins les éléments d'« installation » de l'extension dans l'interface **QGIS** (menu et barre d'outils).



A partir de ce stade, nous allons progresser vers une extension avec interface(s).

Attention aux imports de modules : modules **QtCore** et **QtGui** de **PyQt5**.



QGIS, Créer une extension ...

- Un fichier « *metadata.txt* » (obligatoire et structuré, c'est le descriptif de l'extension : métadonnées). Le formalisme ci-dessus est à respecter. Il peut évoluer avec les versions de QGIS.

Attention aux caractères accentués notamment !!



```
[general]
name = MAQUETTE (Outils)
description=Maquette pour la formation Python 4 QGIS3
category = Tools
version=3.0
qgisMinimumVersion=3.0
# Optional items:
changelog:
=====
** 3.0.0 : Didier LECLERC CMSIG DON
    Version sous Qgis 3
    - Formation Python pour Qgis 3
=====

tags= Tools, Python 4 QGIS3
homepage=http://geo-info.metier.i2
tracker=http://geo-info.metier.i2
repository=http://geo-info.metier.i2
icon=icons/extension.png
experimental=False
deprecated=False

# Author contact information
author=Didier LECLERC - CP2I DO Nord / Hauts de
France, Normandie et Outre-mer atlantique
email=didier.leclerc@developpement-durable.gouv.fr
class_name=TestPlugin
```

Extensions | Installées (17)

Rechercher

- ☒ Auto filtre 3
- ☒ Auto Geom 3
- ☐ DB Manager
- ☐ eVis
- ☐ GRASS 7
- ☐ Géoréférenceur GDAL
- ☒ I.V.E 3 (Eolien) International
- ☐ Layers menu from project
- ☒ MAQUETTE (Outils)
- ☐ MetaSearch Catalog Client
- ☐ Outils GPS
- ☒ Plugin Reloader
- ☐ Processing
- ☐ Saisie de coordonnées
- ☐ Vérificateur de géométrie
- ☐ Vérificateur de topologie
- ☐ Édition hors connexion

MAQUETTE (Outils)

Maquette pour la formation Python 4 QGIS3

Catégorie Tools

Étiquettes Tools, Python 4 QGIS3

Plus d'infos [Page d'accueil](#) [bug tracker](#) [code repository](#)

Auteur Didier LECLERC - CP2I DO Nord / Hauts de France, Normandie et Outre-mer atlantique

Installed version 3.0

Changelog

```
=====
** 3.0.0 : Didier LECLERC CMSIG DON
    Version sous Qgis 3
    - Formation Python pour Qgis 3
=====
```



QGIS, Créer une extension ...

- Un fichier « **`__init__.py`** » (c'est le lanceur de l'extension) ;

Exemple :

```
def classFactory(iface):  
    from .main import MainPlugin  
    return MainPlugin(iface)
```

Appel du constructeur
primaire.

Ce code est générique et réutilisable (avec un module **main** et une classe **MainPlugin**).




QGIS, Créer une extension ...

- Cette classe « MainPlugin » présente dans le module « main » comporte (au moins) deux fonctions (dans le modèle fourni par la suite) :

- « `__init__` » ;
- « `initGUI` ».

Exemple :



```
def __init__(self, iface):  
    self.name = "MonExtension"  
    #référence à l'objet interface QGIS  
    self.iface = iface  
  
def initGui(self):  
    self.menu=QMenu("Mon extension")
```

Les fonctions de déchargement de l'extension (*unload*) ne sont pas indispensables sous QGIS.



QGIS, Créer une extension ...

- La fonction « *initGUI* » de la classe « *MainPlugin* » du module « *main.py* » comprend au moins les éléments d'« installation » de l'extension dans l'interface QGIS (menu, barre d'outils, connections des « actions », séquence de traduction ...).

Le menu est créé avec l'objet **QMenu** de la librairie Qt.

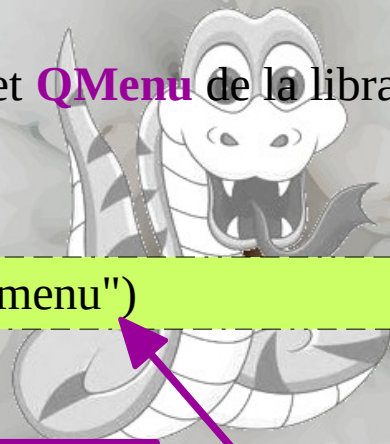
Exemple :

```
self.menu=QMenu("Mon menu")
```

Nom de mon
menu (objet).

Texte affiché dans la barre
de menus.

Référence à mon objet
(classe) extension.



QGIS, Créer une extension ...

Les différentes « actions » (*lignes de commandes*) contenues dans mon menu sont créées avec l'objet **QAction** de la librairie **Qt**.

Exemple :

```
self.commande1 = QAction(QIcon(menuIcon),"Ligne de commande 1  
...self.iface.mainWindow())
```

Nom de l'action
(objet).

Chemin du pictogramme pour
l'action

Référence à la fenêtre
principale de l'interface QGIS.

Texte affiché dans la barre
du menu pour l'action.

Référence à mon objet
(classe) MainPlugin.



QGIS, Créer une extension ...

Pour associer une action à un menu, rien de plus simple :

Exemple :

```
self.menu.addAction(self.commande1)
```

Nom du menu
(objet).

Nom de l'action
(objet).

Référence à mon objet
(classe) extension.



QGIS, Créer une extension ...

Pour associer une action à un menu, rien de plus simple :

Exemple

Nom de l'action
(objet).

Signal envoyé
par l'objet.

```
self.commande5.triggered.connect(self.LoadDlgBoxQt)
```

Connexion de la commande
à l'action

Une fonction (slot)
particulière

QGIS, Créer une extension ...

Pour associer une action à un menu, rien de plus simple :

Exemple :

```
menuBar = self.iface.mainWindow().menuBar()  
menuBar.addMenu(self.menu)
```

Instancie la barre de menu

Ajoute mon menu à la barre de Menu





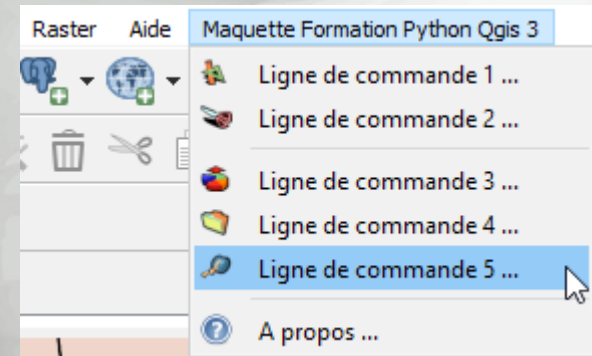
QGIS, Créer une extension ...

Pour résumé un petit schéma ..., ça vaut mieux que des grandes phrases :

Démonstration

Qu'est ce que nous voulons obtenir ?

- Un menu dans une barre de menu
- Des items de menu avec des icônes personnalisées
- Des séparateurs d'items
- Des actions pour chaque items sélectionnés





QGIS, Créer une extension ...

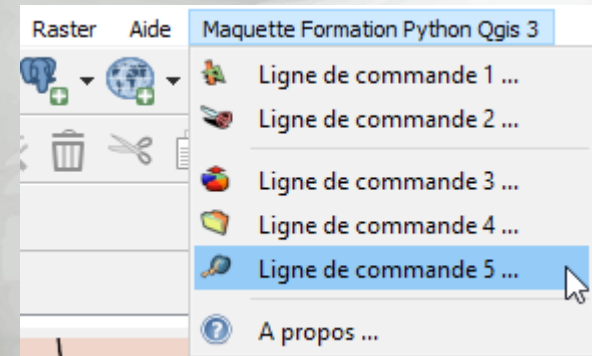
Pour résumé un petit schéma ..., ça vaut mieux que des grandes phrases :

Démonstration

__init__.py

```
def classFactory(iface):  
    from ,main import MainPlugin  
    return MainPlugin(iface)
```

Le lanceur





QGIS, Créer une extension ...

Pour résumé un petit schéma ..., ça vaut mieux que des grandes phrases :

Démonstration

__init__.py

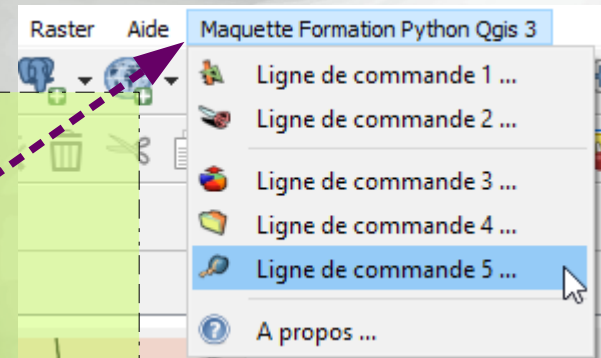
```
def classFactory(iface):  
    from ,main import MainPlugin  
    return MainPlugin(iface)
```

main.py

```
class MainPlugin(object):  
    def __init__(self, iface):  
        #référence à l'objet interface QGIS  
        self.iface = iface  
  
    def initGui(self):  
        self.menu=QMenu("Maquette Formation Python Qgis 3")  
        ...
```

Le fichier main

La classe MainPlugin
et ses deux fonctions





QGIS, Créer une extension ...

Pour résumé un petit schéma ..., ça vaut mieux que des grandes phrases :

Démonstration

__init__.py

```
def classFactory(iface):  
    from ,main import MainPlugin  
    return MainPlugin(iface)
```

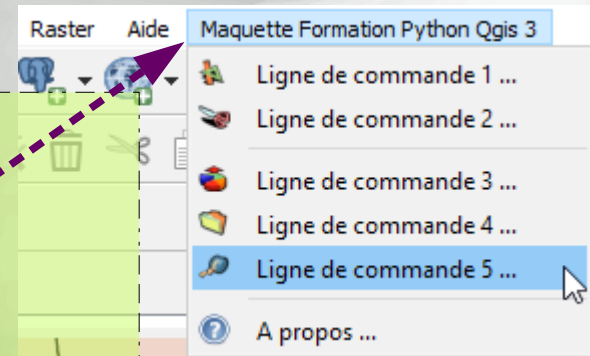
main.py

```
class MainPlugin(object):  
    def __init__(self, iface):  
        #référence à l'objet interface QGIS  
        self.iface = iface  
  
    def initGui(self):  
        self.menu=QMenu("Maquette Formation Python Qgis 3")  
        ...
```

Zoom sur initGui

def initGui(self):

```
self.menu=QMenu("Maquette Formation Python Qgis 3")  
  
#déclaration des actions élémentaires  
menuIcon = "exemple1.png"  
self.commande1 = QAction(QIcon(menuIcon),"Ligne de commande 1, self.iface.mainWindow())  
menuIcon = "exemple2.png"  
self.commande2 = QAction(QIcon(menuIcon),"Ligne de commande 2, self.iface.mainWindow())
```





QGIS, Créer une extension ...

Pour résumé un petit schéma ..., ça vaut mieux que des grandes phrases :

Démonstration

__init__.py

```
def classFactory(iface):  
    from ,main import MainPlugin  
    return MainPlugin(iface)
```

main.py

```
class MainPlugin(object):  
    def __init__(self, iface):  
        #référence à l'objet interface QGIS  
        self.iface = iface
```

```
    def initGui(self):  
        self.menu=QMenu("Maquette Formation Python Qgis 3")  
        ...
```

Zoom sur initGui

def initGui(self):

```
    self.menu=QMenu("Maquette Formation Python Qgis 3")
```

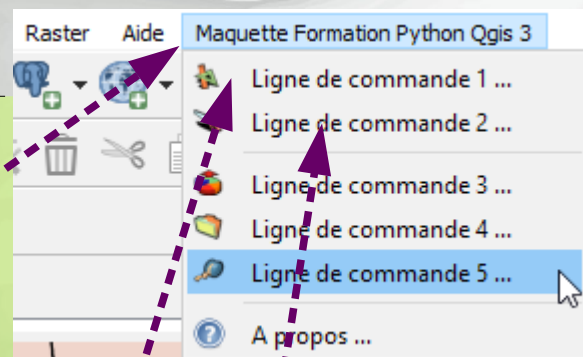
```
    #déclaration des actions élémentaires
```

```
    menuIcon = "exemple1.png"
```

```
    self.commande1 = QAction(QIcon(menuIcon),"Ligne de commande 1, self.iface.mainWindow())
```

```
    menuIcon = "exemple2.png"
```

```
    self.commande2 = QAction(QIcon(menuIcon),"Ligne de commande 2, self.iface.mainWindow())
```



```
#Construction du menu  
self.menu.addAction(self.commande1)  
self.menu.addAction(self.commande2)  
self.menu.addSeparator()
```





QGIS, Créer une extension ...

Pour résumé un petit schéma ..., ça vaut mieux que des grandes phrases :

Démonstration

__init__.py

```
def classFactory(iface):  
    from ,main import MainPlugin  
    return MainPlugin(iface)
```

main.py

```
class MainPlugin(object):  
    def __init__(self, iface):  
        #référence à l'objet interface QGIS  
        self.iface = iface
```

```
    def initGui(self):  
        self.menu=QMenu("Maquette Formation Python Qgis 3")  
        ...
```

Zoom sur initGui

def initGui(self):

```
    self.menu=QMenu("Maquette Formation Python Qgis 3")
```

```
    #déclaration des actions élémentaires
```

```
    menuIcon = "exemple1.png"
```

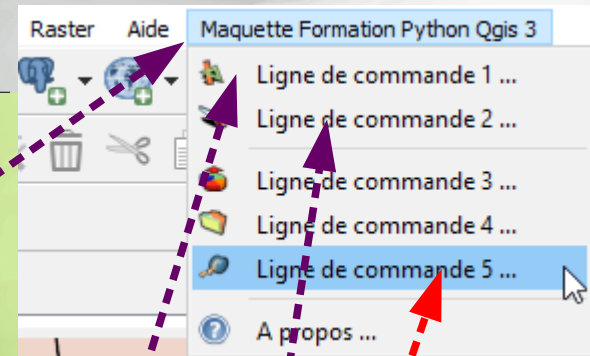
```
    self.commande1 = QAction(QIcon(menuIcon),"Ligne de commande 1, self.iface.mainWindow())
```

```
    menuIcon = "exemple2.png"
```

```
    self.commande2 = QAction(QIcon(menuIcon),"Ligne de commande 2, self.iface.mainWindow())
```

```
    #Connection de la commande à l'action
```

```
    ** self.commande5.triggered.connect(self.LoadDlgBoxQt)
```



```
#Construction du menu  
self.menu.addAction(self.commande1)  
self.menu.addAction(self.commande2)  
self.menu.addSeparator()
```

Déclencheur, sur commande5





QGIS, Créer une extension ...

Pour résumé un petit schéma ..., ça vaut mieux que des grandes phrases :

Démonstration

`__init__.py`

```
def classFactory(iface):  
    from ,main import MainPlugin  
    return MainPlugin(iface)
```

`main.py`

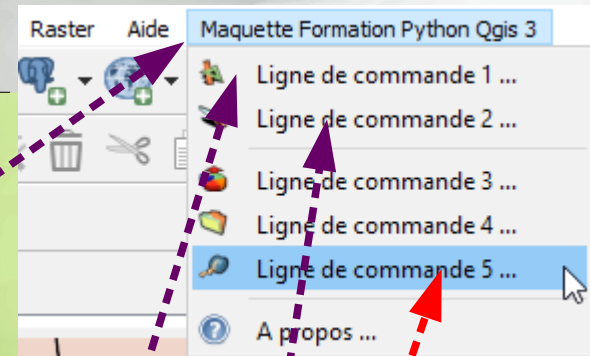
```
class MainPlugin(object):  
    def __init__(self, iface):  
        #référence à l'objet interface QGIS  
        self.iface = iface
```

```
    def initGui(self):  
        self.menu=QMenu("Maquette Formation Python Qgis 3")  
        ...
```

Zoom sur initGui

`def initGui(self):`

```
    self.menu=QMenu("Maquette Formation Python Qgis 3")  
  
    #déclaration des actions élémentaires  
    menuIcon = "exemple1.png"  
    self.commande1 = QAction(QIcon(menuIcon),"Ligne de commande 1, self.iface.mainWindow())  
    menuIcon = "exemple2.png"  
    self.commande2 = QAction(QIcon(menuIcon),"Ligne de commande 2, self.iface.mainWindow())
```



```
    #Construction du menu  
    self.menu.addAction(self.commande1)  
    self.menu.addAction(self.commande2)  
    self.menu.addSeparator()
```

Déclencheur, sur commande5

Ouf, j'ajoute le menu

```
menuBar = self.iface.mainWindow().menuBar()  
menuBar.addMenu(self.menu)
```

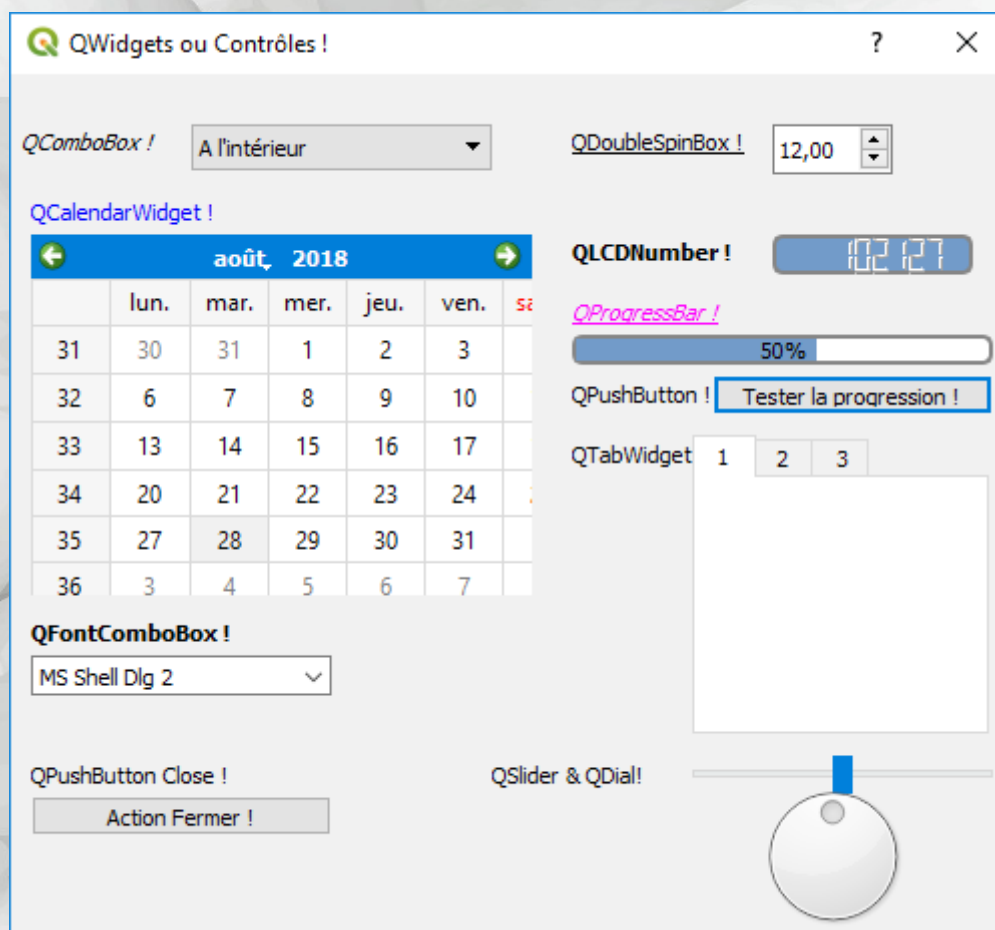
```
#Connection de la commande à l'action  
** self.commande5.triggered.connect(self.LoadDlgBoxQt)
```



QGIS, Créer une extension ...

Au-delà des éléments de menu, une extension peut comporter une ou des boîtes de dialogue composée(s) d'objets (contrôles ou Widgets).

Le modèle d'extension que l'on va utiliser par la suite vous fournit cette boîte de dialogue avec ces exemples de widgets.





QGIS, Créer une extension ...

Un kit de démarrage est à votre disposition avec un modèle d'extension :

- **Modèle :**

Modèle simple avec menu et présentation objets Qt sur la « commande 5 ».

Introspection du code

- **Modèle (niveau 1) :**

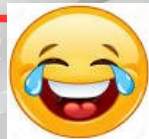


Mais où dois-je mettre ce module et l'ensemble de ses éléments ?

- **Modèle (niveau 2) :**

Réponse : au bon endroit

Arrêtons de rire



Vous aurez oublié que :

Version Ministère : Pas d'infos à ce jour

Version Communautaire : C:\Users\MonProfilAMoi\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins





QGIS, Créer une extension ...

Exercices

A partir du modèle, créer le niveau 1 :

- Modèle :

Modèle simple avec menu et présentation objets Qt sur la « commande 5 ».

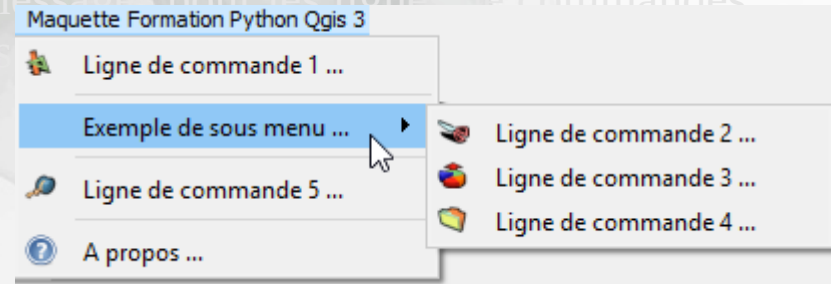
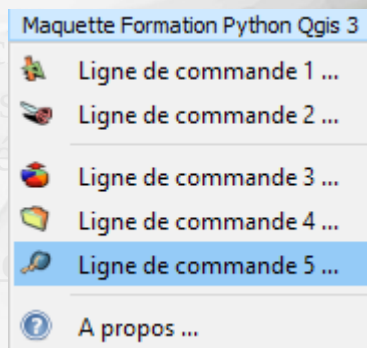
- Modèle (niveau 1) :

Enrichissement du menu avec un sous-menu.

- Modèle (niveau 2) :

Enrichissement avec une boîte de dialogue présentant les métadonnées de la couche sélectionnée dans une liste déroulante.

- Modèle



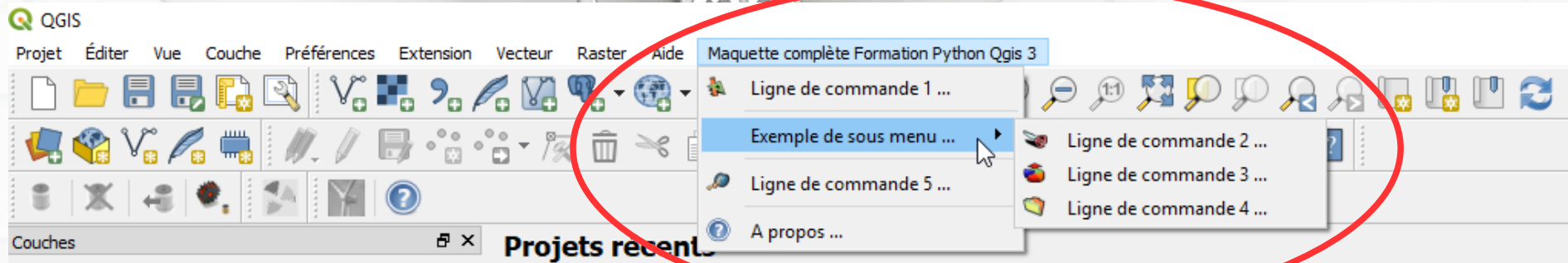


QGIS, Créer une extension ...



Comment personnaliser le positionnement de mon menu ?

Mais non, je ne parle pas de menu gastronomique !!, mais de la position du menu qui lance votre plugin



Démonstration



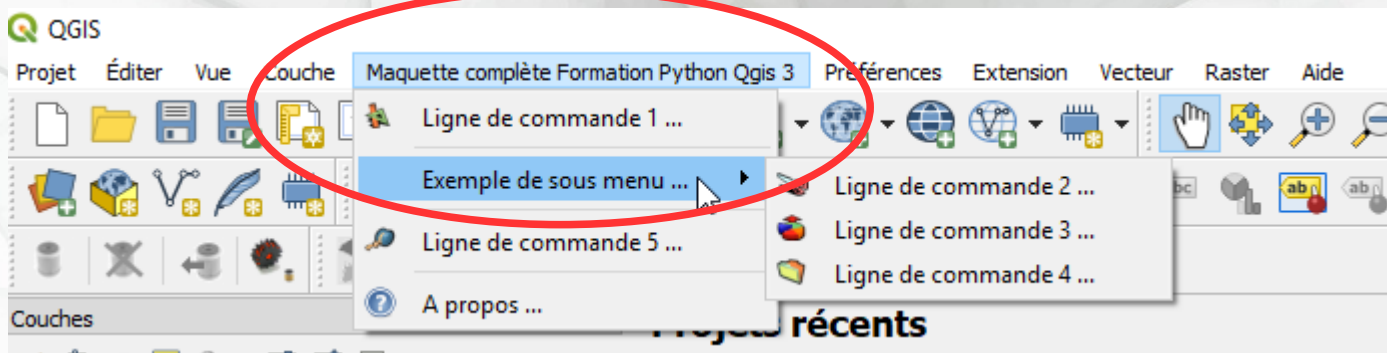


QGIS, Créer une extension ...

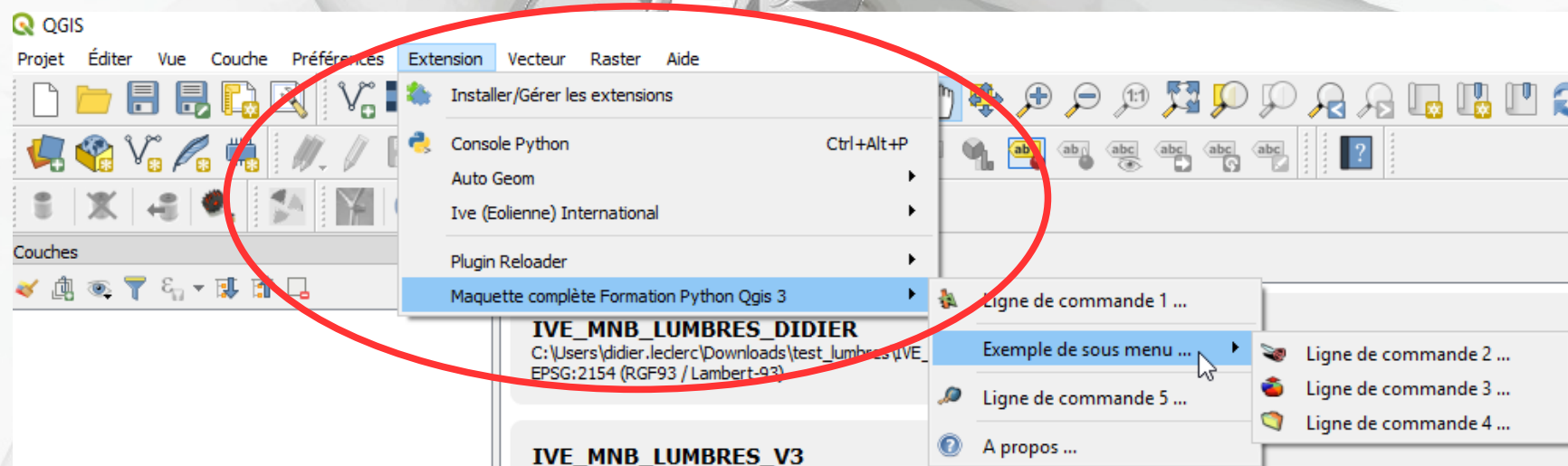


Comment personnaliser le positionnement de mon menu

- Soit dans la barre de menu

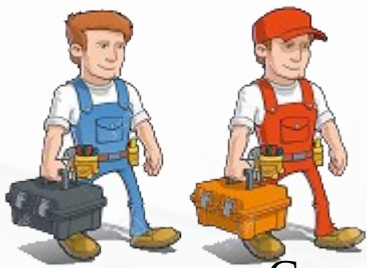


- Soit dans le menu Extension de la barre de menu



Démonstration





QGIS, Créer une extension ...



Comment personnaliser le positionnement de mon menu
A la place de :

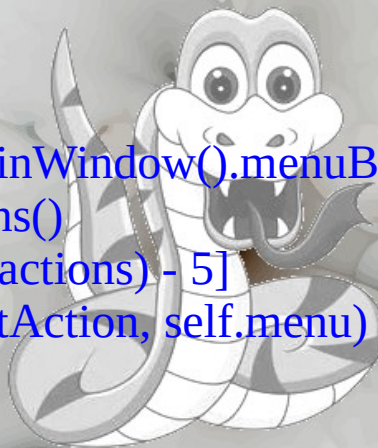
```
menuBar = self.iface.mainWindow().menuBar()  
menuBar.addMenu(self.menu)
```

Mettre

```
menuBar = self.iface.mainWindow().menuBar()  
actions = menuBar.actions()  
lastAction = actions[len(actions) - 5]  
menuBar.insertMenu(lastAction, self.menu)
```

Ou Bien mettre

```
menuBar = self.iface.mainWindow().menuBar()  
zMenu = menuBar  
for child in menuBar.children():  
    if child.objectName() == "mPluginMenu" :  
        zMenu = child  
        break  
zMenu.addMenu(self.menu)
```



Démonstration



QGIS, Créer une extension ...

Une action peut aussi être assignée à une **barre d'outils** (passage par la fonction **addToolBar** de la classe **QgisInterface**).

Exemple :

```
self.toolbarName = "Ma Barre"  
self.toolbar = self.iface.addToolBar(self.toolbarName)
```

```
self.toolbar.addAction(self.commande1)
```





QGIS, Créer une extension ...

Exercices

A partir du modèle, créer le niveau 1 :

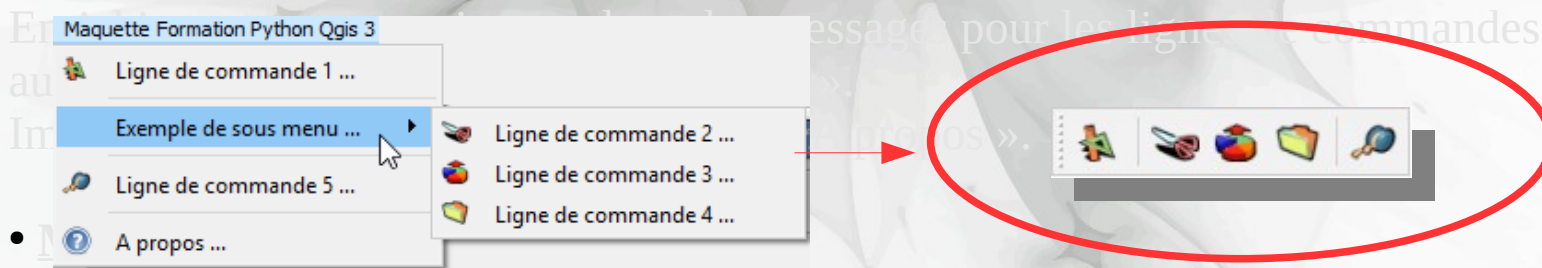
- Modèle :

Modèle simple avec menu et présentation objets Qt sur la « commande 5 ».

- Modèle (niveau 1) :

Enrichissement du menu avec une barre d'outils.

- Modèle (niveau 2) :



Enrichissement avec une boîte de dialogue présentant les métadonnées de la couche sélectionnée dans une liste déroulante.



QGIS, Créer une extension ...

La librairie **Qt** fournit aussi une classe pour l'affichage de messages avec différentes symboliques.

QMessageBox.information(None,"Information :", "Je suis un message d'information.\nEn format texte classique.")

QMessageBox.information(None,"Information :", "Je suis un message d'information.
En format HTML
<hr>Notez la puissance de feu.")

QMessageBox.warning(None,"Avertissement :", "Je suis un message d'alerte.\nEn format texte classique.")

QMessageBox.critical(None,"Alerte :", "Je suis un message de cas critique.
En format HTML.")

QMessageBox.question(None, "Confirmation", "Je suis une demande de confirmation.
Confirmez-vous le lancement de l'opération :
ESSAI
pour la couche :
<u>[TEST]</u>",QMessageBox.Yes|QMessageBox.No)

Notez que les formats HTML ou texte standard sont supportés, et que le type (« **information** », « **warning** », « **critical** » ou « **question** » précise le pictogramme affiché dans la boîte de dialogue).



QGIS, Créer une extension ...

Les widgets ont tous des fonctions (publiques) spécifiques mais disposent d'un noyau dur de fonctions (héritage objet de la classe de base **QWidget** ou de la classe **QObject**, dont elle-même hérite) :

Parmi ce noyau dur, quelques fonctions de base pour instancier un objet « widget » :

Exemple :

Nom de
mon objet
contrôle

TypeWidget : exemple de valeurs

- QLabel ;
- QPushButton ;
- QSlider ;
- QProgressBar ;
-

```
self.MonWidget = QtWidgets.TypeWidget(Dialog)
self.MonWidget.setMinimumSize(QtCore.QSize(150, 20))
self.MonWidget.setMaximumSize(QtCore.QSize(150, 20))
self.MonWidget.setGeometry(QtCore.QRect(10, 360, 150, 20))
self.MonWidget.setObjectName("MonWidget")
```



Attention aux nouvelles déclarations pour PyQt5

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtCore import *
from PyQt5.QtWidgets import QAction, QMenu, QApplication
```

Toutes ces fonctions sont
génériques (dimension,
nom du contrôle)



QGIS, Connecter des actions ...

Les contrôles ou widgets de nos boîtes de dialogue vont réagir à partir d'événements qu'il va falloir gérer.

On parle de **signaux** et de **slots**. Ils sont une implémentation du patron de conception observateur utilisée par la bibliothèque logicielle **Qt**. Ils se caractérisent ainsi :

- Un **signal** : c'est un message envoyé par un contrôle lorsqu'un événement se produit.

Exemple : on a cliqué sur un bouton.

- Un **slot** : c'est la fonction qui est appelée lorsqu'un événement s'est produit. On dit que le signal appelle le slot. Concrètement, un slot est une fonction d'une classe.

Exemple : le slot **quit()** de la classe **QApplication**, qui provoque l'arrêt du programme.

Les signaux et les slots peuvent être vus comme des attributs et des fonctions d'une classe.



QGIS, Connecter des actions ...

La syntaxe générique pour connecter un objet à une action est la suivante :

objet."type".connect(fonction)

L'objet qui émet
le signal "type"

appelle

fonction

Exemples de type de signal :

"**clicked**" [QPushButton, QTreeView, QLabel ...]

"**currentIndexChanged**" [QComboBox, QListView, ...]

"**doubleClicked**" [QTreeView, QListView, ...]

"**textChanged**" [QTextEdit, ...]

"**valueChanged**" [QSlider, ...]

"**lastWindowClosed**"

"**timeout**"

...



Attention à la syntaxe et à l'utilisation des signaux des objets

Dans l'attente de l'implémentation de la documentation de PyQt5, consulter celle de QT5.



QGIS, Connecter des actions ...

Des actions génériques :

Exemple :

```
self.CloseButton.clicked.connect(Dialog.reject)
```

L'objet qui émet
le signal

appelle

Une fonction (slot)
particulière

Rappel :
Attention, 'self.CloseButton' et 'CloseButton '
ne font pas référence au même objet

Dans le cas d'une boîte de dialogue, au moins trois slots génériques peuvent être employés : « **accept** », « **reject** » et « **close** ».
Ces deux premiers slots permettent de tracer (éventuellement) si la boîte de dialogue a été "acceptée" ou non.

Les éléments saisis dans les différents contrôles ne seront pas réinitialisés dans les deux cas. Il est tout à fait possible de personnaliser le comportement du programme sur ces slots.

Exemple :

```
def reject(self):  
    self.SaveInfosIni()  
    self.hide()
```

Fonction d'une
classe (appel
d'une fonction de
sauvegarde)



QGIS, Connecter des actions ...

La passage d'ordre entre contrôles se décrit de la manière suivante :

```
QtCore.QMetaObject.connectSlotsByName(Dialog)
Dialog.setTabOrder(self.control1,self.control2)
Dialog.setTabOrder(self.control2,self.control3)
Dialog.setTabOrder(self.control3,self.control4)
...
Dialog.setTabOrder(self.controln-1,self.controln)
```

Objet conteneur (self,
dialog, ...)

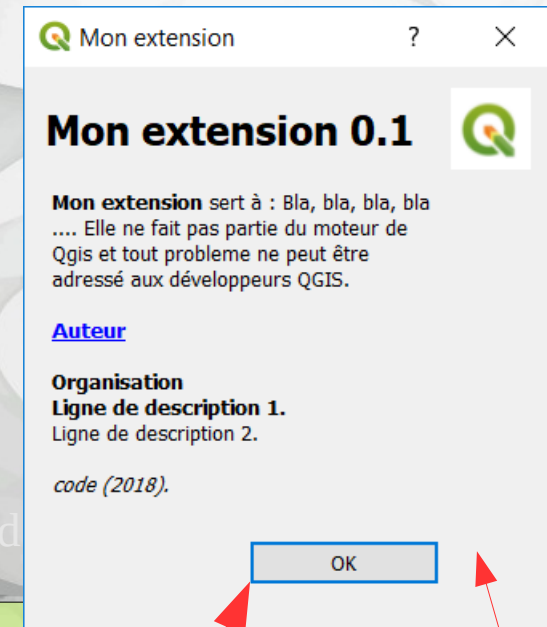
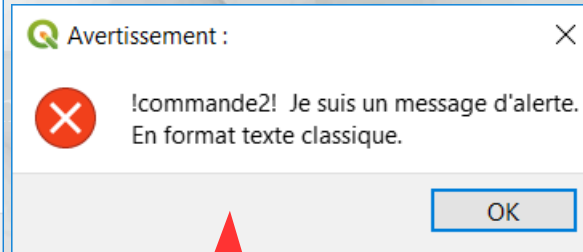
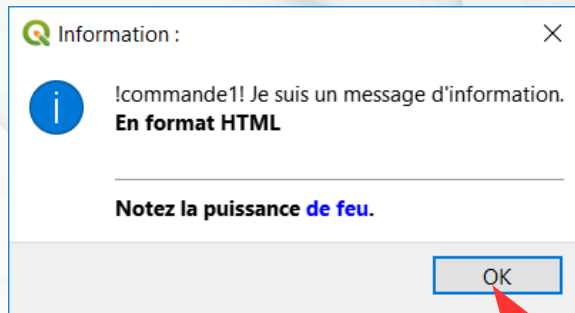
Pas indispensable si les objets sont implémentés dans le bon « ordre » dans le code Python.





QGIS, Connecter des actions ...

A partir du modèle niveau 1, créer le niveau 2 :



Enrichissement du modèle avec un sous-menu et une barre d'outils

- **Modèle (niveau 2) :**

- Deux tâches à réaliser

- ** Enrichissement avec mise en place des messages pour les lignes de commandes autres que « commande 5 » et « A propos ».
- ** Implémentation de la boîte de dialogue « A propos ».

(boîte fournie dans le répertoire « niv2 »)

- **Modèle (niveau 3) :**

Enrichissement avec une boîte de dialogue présentant les métadonnées de la





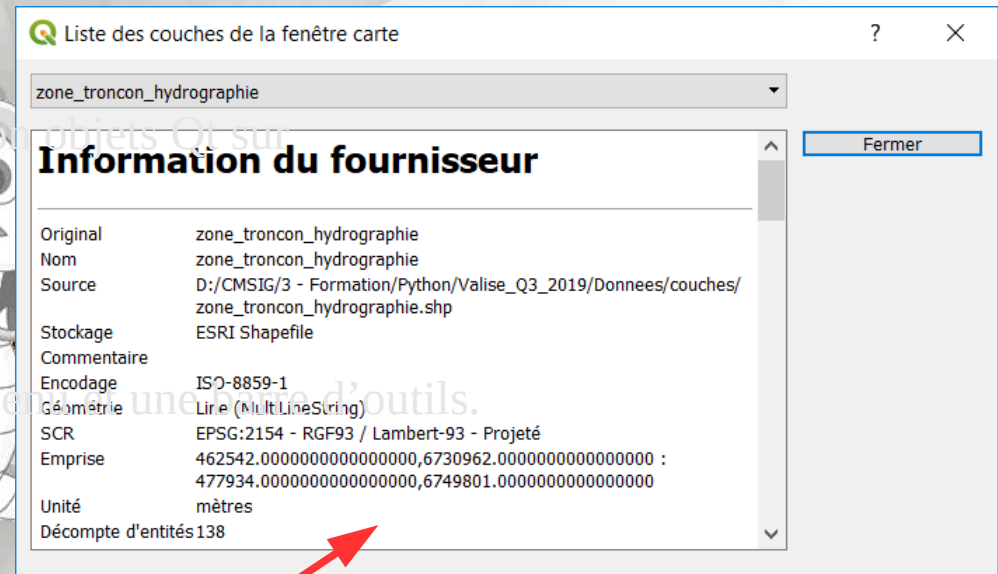
QGIS, Connecter des actions ...

A partir du modèle niveau 2, créer le niveau 3 :

Chargez le projet « **test_QGIS3.qgs** ».

La nouvelle boîte de dialogue, qui vous est fournie dans le répertoire niv3 est à implémenter et à compléter.

- Elle sera appelée par l'action « **commande1** » ;
- Elle permettra de sélectionner une couche du projet et d'afficher les métadonnées de cette couche ;
- Pour la fermer, une confirmation sera nécessaire (fournie dans la maquette).



• Modèle (niveau 3) :

Enrichissement avec une boîte de dialogue présentant les métadonnées de la couche sélectionnée dans une liste déroulante.

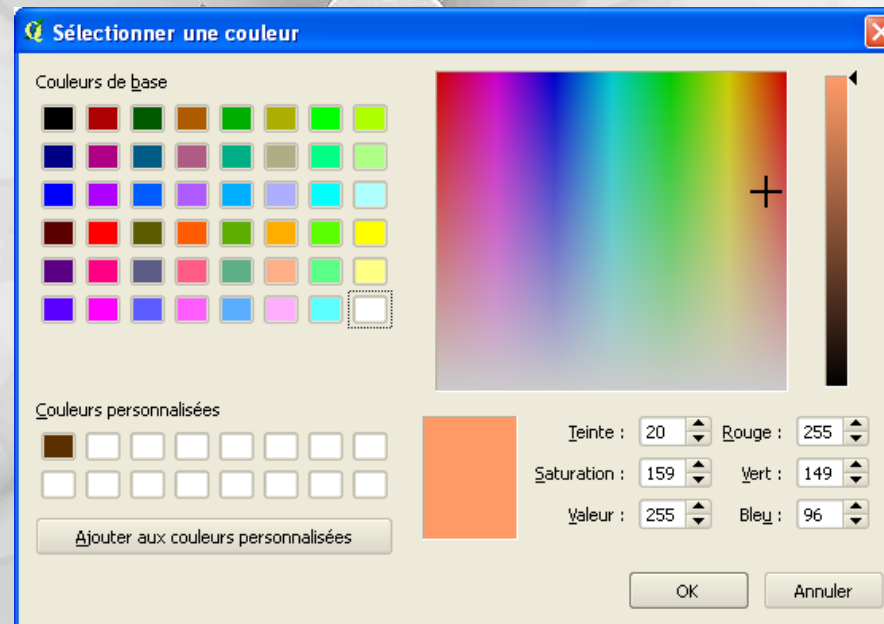


QGIS, Créer une extension ...

On l'a vu, la librairie Qt fournit une richesse d'objets (contrôles ou Widgets) pour réaliser des interfaces conviviales.

Exemples :

QcolorDialog



QFileDialog (exemple fourni dans la rubrique « *Allez plus loin avec QGIS* »)

QPrinter (exemple fourni dans la rubrique « *Allez plus loin avec QGIS* »)

QGIS, Connecter des actions ...

Utiliser les boîtes de dialogue Fichiers et Répertoires :

Dans la construction d'un programme, il est souvent nécessaire de l'interfacer avec son environnement (chargement, sauvegarde de fichiers, sélection d'un répertoire de travail ...). La classe **QFileDialog** de la librairie **Qt** répond parfaitement à ces besoins.

Exemples :

```
InitDir = os.path.dirname(__file__)  
fileName = QFileDialog.getOpenFileName(None, "Fichier Shape :", InitDir, "*.shp")  
inputDir = QFileDialog.getExistingDirectory(self, "Sélectionner un dossier", InitDir, QFileDialog.ShowDirsOnly)
```

Initialisation du répertoire courant
(**__file__** : là où le fichier py exécuté)

Sélection fichier (Shp) ;
Sélection dossier.

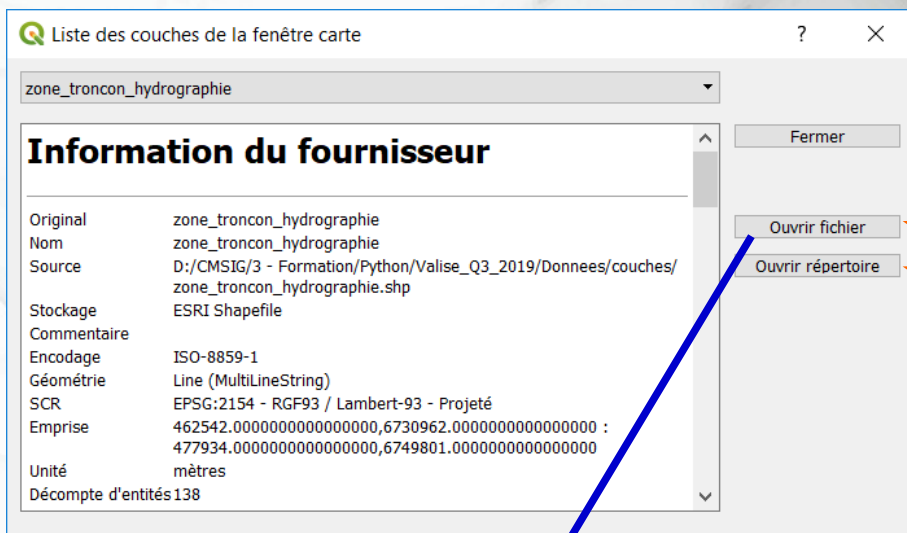
Options pour
restreindre le contenu
restitué.





QGIS, Connecter des actions ...

Exercices

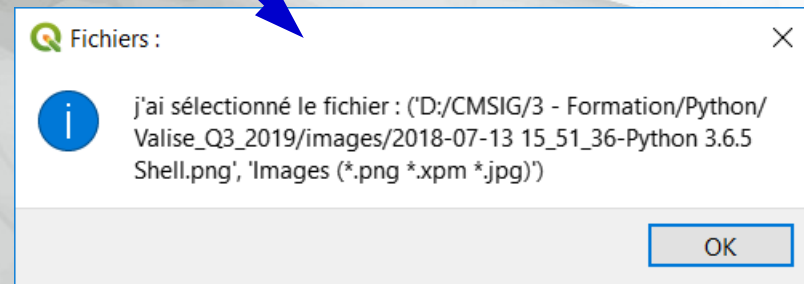
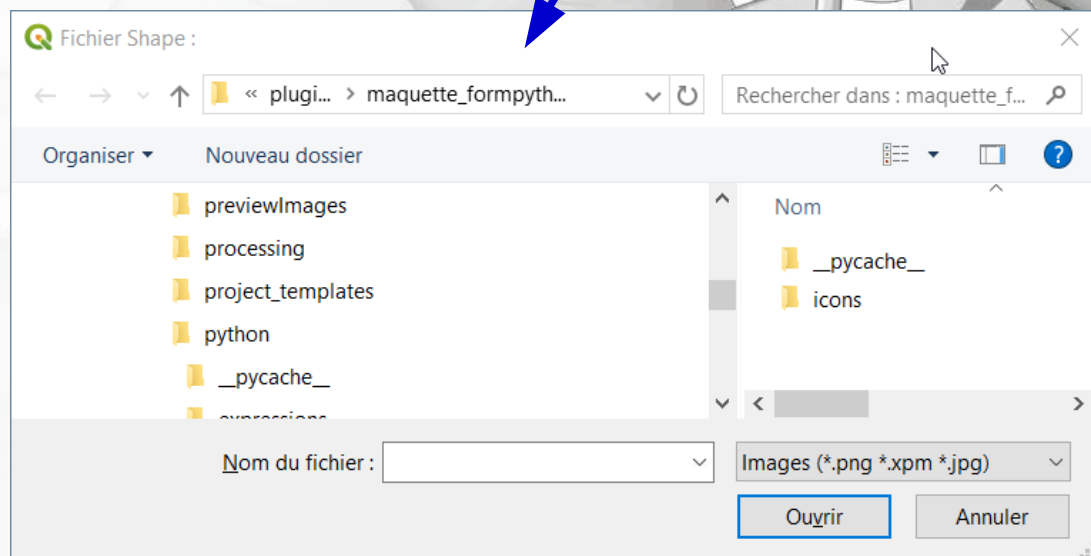


Enrichir encore **Modèle (niveau 4)** :

Enrichissement de la boîte de dialogue des méta-données

avec deux boutons :

- Le premier pour ouvrir des fichiers ;
- Le second pour ouvrir des répertoires ;
- Merci de confirmer les choix dans chaque boîte de dialogue par un message d'information.



QGIS, Connecter des actions ...

Imprimer :

Qt5 propose des classes dédiées à la réalisation d'impression : **QPrinter()**, **QPrinterDialog()**, **QPrintPreviewDialog()** du module QtPrintSupport ...

```
def printMetadatLayer(self):  
    printer = QPrinter()  
    printer.setPageSize(QPrinter.A4)  
    printer.setOrientation(QPrinter.Portrait)  
    printer.setOutputFormat(QPrinter.NativeFormat)  
    printDialog = QPrintPreviewDialog(printer)  
    zTitle = "Imprimer les métadonnées de la couche"  
    printDialog.setWindowTitle(zTitle)  
    printDialog.setWindowState(Qt.WindowNoState)  
    printDialog.paintRequested.connect(self.TextEdit.print_)  
    printDialog.exec_()
```

Initialisation d'une instance de la classe (cf. constructeur)

Contenu à imprimer (ici, le contenu d'un widget)



Attention aux imports depuis la version PyQt5

```
from PyQt5.QtPrintSupport import QPrintDialog, QPrinter, QPrintPreviewDialog
```

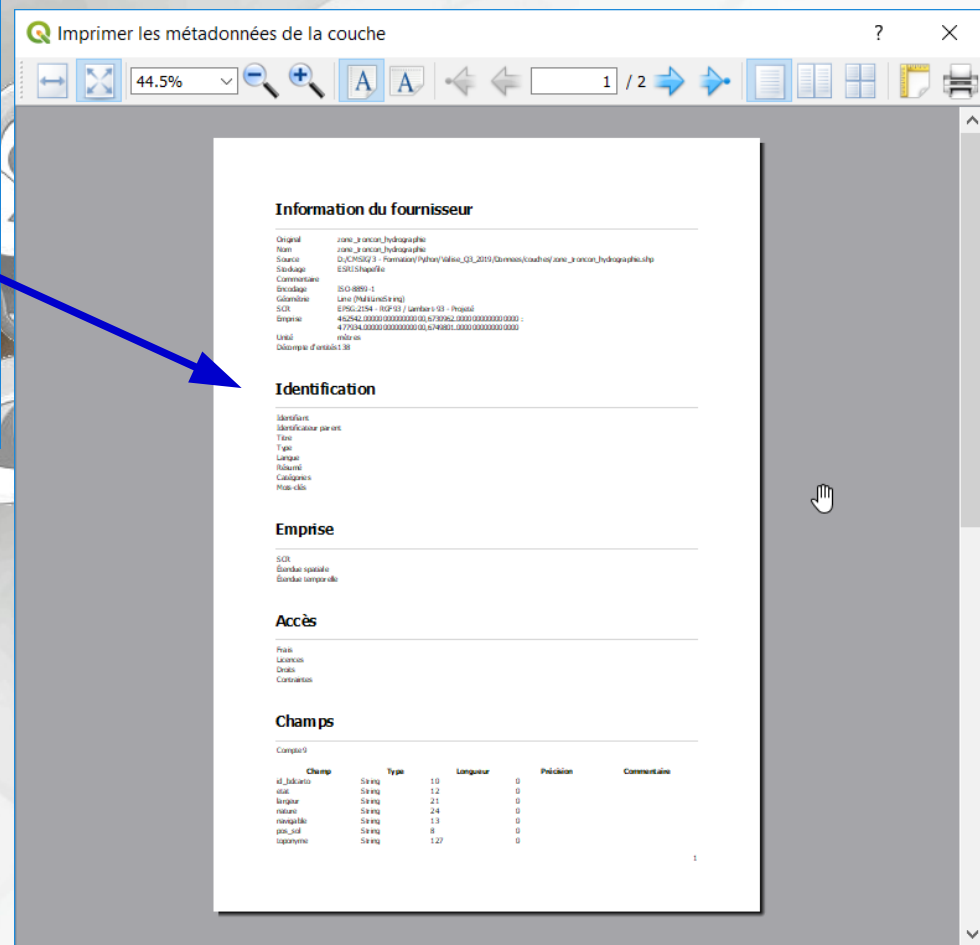
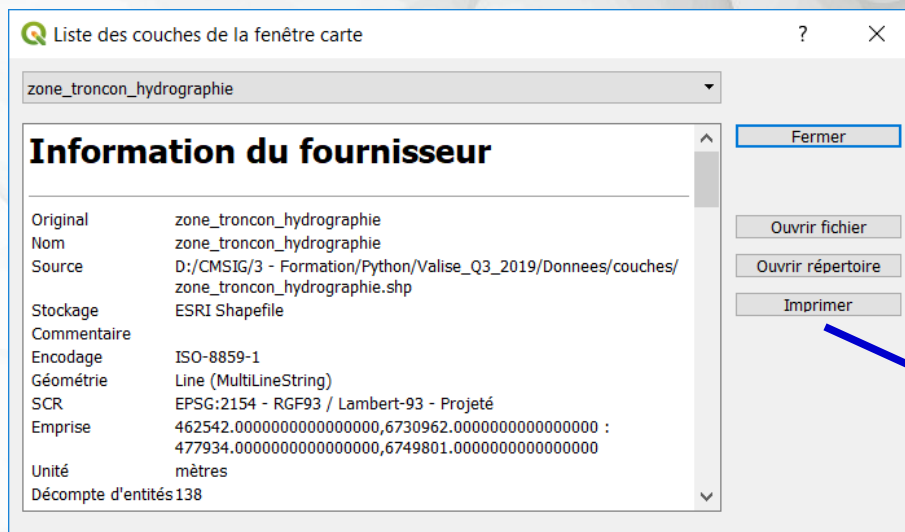




QGIS, Connecter des actions ...

Imprimer :

Exercices



QGIS, Manipuler des couches ...

Ici, il ne sera question que de couches vecteurs. Pour les raster, des classes et des fonctions sont dédiées (cf. **QgsRasterLayer** de l'API).

Concepts

QgsVectorLayer
zLayer = ...

Il nous faut pointer sur un objet vecteur.

`Self.iface.mapCanvas().addVectorLayer (PathOfTheLayer, NameForTheLayer, 'ogr')`

`Self.iface.mapCanvas().currentLayer()` - ATTENTION ou - `Self.iface.activeLayer()`

`Self.iface.mapCanvas().layer(i)`

`QgsVectorLayer("Polygon", "Ma couche", "memory")`

Constructeur
de la classe

Fields()

Geometry()

Contenu attributaire.

Contenu objets géographiques.



QGIS, Manipuler des couches ...

Concepts

Pour les couches « mémoires » créées à l'aide du constructeur, deux informations sont attendues par ce constructeur :

```
QgsVectorLayer("Polygon", "Ma couche", "memory")
```

Le type d'objet :

- Point ;
- LineString ;
- Polygon.

Le nom de cette nouvelle couche

Le constructeur permet aussi de créer de nouvelles couches physiquement, quatre informations sont alors attendues par ce constructeur :

```
QgsVectorLayer(zPathLayerl, zNameLayer, zNameProvider, zLoadDefaultStyleFlag)
```

Le chemin du fichier pour cette nouvelle couche

Le nom de cette nouvelle couche

Le « **Provider** » de cette nouvelle couche

Flag qui indique si un style à défaut est à appliquer à la couche (True ou False)

QGIS, Manipuler des couches ...

Concepts

Le PROVIDER



QGIS, Manipuler des couches ...



Le « **provider** » est un concept important. C'est le pilote ou le fournisseur pour les données (format) de la couche vecteur. Pour notre objet « **zLayer** », nous pouvons obtenir un pointeur sur le **provider** et son type par les fonctions suivantes :

```
zLayer.dataProvider()  
zLayer.providerType()
```

Une fois l'objet provider défini, le code pour parcourir les enregistrements est du type :

```
zProvider = zLayer.dataProvider()
```

```
allAttrs = zProvider.attributeIndexes() ← La liste des index des attributs
```

```
zFields = zProvider.fields() ← Pointeur objet sur les champs
```

```
feat = QgsFeature() ← Pointeur enregistrements
```



QGIS, Manipuler des couches ...



Le PROVIDER

```
zProvider = zLayer.dataProvider()
```

```
#Exemple de boucle sur les enregistrements
```

```
MyAllValues = MyActiveLayer.getFeatures(QgsFeatureRequest())
```

```
for MyValue in MyAllValues:
```

```
    #Renvoi un Tuple contenant l'ensemble des valeurs
```

```
    print(str(MyValue.attributes()))
```

```
#Juste un attribut
```

```
MyAttribIndex = 2
```

```
for MyValue in MyAllValues:
```

```
    print(str(MyValue[MyAttribIndex]))
```

```
#Exemple de boucle pour lire la structure attributaire
```

```
zFields = zProvider.fields()
```

```
for field in zFields :
```

```
    zNameField = field.name()
```

cf. classe **QgsField** !

QGIS, Manipuler des couches ...

Ci-dessous un **exemple** pour parcourir les données attributaires d'une couche sur un attribut et obtenir les min et max de l'attribut.

```
def GetMinMax(zLayer, zIndexField):
    zProvider = zLayer.dataProvider()
    minVal, maxVal = 0, 0
    first = True

    for feat in zLayer.getFeatures(QgsFeatureRequest()):
        value = float(feat[ zIndexField ])
        if first: minVal, maxVal, first = value, value, False
        else:
            if value < minVal: minVal = value
            if value > maxVal: maxVal = value

    return (minVal, maxVal)

zLayer = qgis.utils.iface.activeLayer()
zMin, zMax = GetMinMax(zLayer, 3)
print(zMin)
print(zMax)
```

Ce n'est qu'un exemple !

La classe **QgsVectorLayer** dispose des fonctions :

- **minimumValue** (int index)
- **maximumValue** (int index)

Le retour de ces fonctions est de type **Qvariant**.



QGIS, Manipuler des couches ...

Manipulations des enregistrements dans le cas d'une sélection d'objets

```
#Une sélection existe  
zFeatures = zLayer.selectedFeatures()  
  
#On prend tous les enregistrements  
zFeatures = zLayer.getFeatures(QgsFeatureRequest())  
  
for feat in zFeatures :  
    #Action ...
```





QGIS, Manipuler des couches ...

Exercices

Cadeaux

Faites vous plaisir



Afficher :

- Le nombre d'enregistrements
- Les index des attributs
- La définition des attributs
- Les valeurs des attributs

Attention

Il ne manque pas quelque chose ?



```
def Traitement_Couches():
```

```
.....  
.....
```

```
#Provider
```

```
# le conteneur de format
```

```
MyProvider = MyActiveLayer.dataProvider()
```

```
#Nombre d'enregistrements
```

```
print(MyProvider.featureCount())
```

```
# les index des attributs
```

```
MyAttributsIndex = MyProvider.attributeIndexes()
```

```
print(MyAttributsIndex)
```

```
#Mes attributs
```

```
MyAttributs = MyProvider.fields()
```

```
for MyAttrib in MyAttributs:
```

```
    print(str(MyAttrib.name()) + " " + str(MyAttrib.typeName()) + " " +  
          str(MyAttrib.type()) + " " + str(MyAttrib.length()))
```

```
#Mes valeurs
```

```
MyAllValues = MyActiveLayer.getFeatures(QgsFeatureRequest())
```

```
for MyValue in MyAllValues:
```

```
    print(str(MyValue.attributes()))  
    print(MyValue.geometry())
```

```
#Lance le traitement sur les couches
```

```
Traitement_Couches()
```



QGIS, Aller plus loin ...

Qgis toujours plus ...pour les curieux

Connecter une base PostGIS :

QGIS Offre une interface forte avec le serveur de base de données PostgreSQL. Une fois la couche PostGIS chargée, elle se manipule comme n'importe quelle couche vecteur (ou raster, cf. PosGIS 2.0) sous QGIS.

Exemples :

```
uri = QgsDataSourceURI()  
# établir la connection sur la base  
uri.setConnection("localhost", "5434", "service", "postgres", "postgres")  
# sélectionner une source de données, avec ou sans critère  
uri.setDataSource("donnees", "communes_colmar_l93", "geom", "")  
# Instancie la variable vlayer avec l'objet de type QgsVectorLayer  
vlayer = QgsVectorLayer(uri.uri(), "ma couche PostGIS", "postgres")  
# Charge dans le CanVas la couche vlayer  
vlayer = QgsProject.instance().addMapLayers([vlayer])  
  
uri.setDataSource("donnees", "communes_colmar_l93", "geom", "popu >= 1000")  
vlayer = QgsVectorLayer(uri.uri(), "ma couche PostGIS FILTRER", "postgres")  
vlayer = QgsProject.instance().addMapLayers([vlayer])
```

Initialisation d'une instance de la classe (cf. constructeur)

Critère pour restreindre le contenu restitué.



QGIS, Aller plus loin ...

Qgis toujours plus ...pour les curieux

Utiliser la boîte à outils « Traitements » :

Les différentes bibliothèques de la boîte à outils sont accessibles depuis le menu « **Traitements** ». Il est possible de construire des modèles, de réaliser des traitements par lot, d'ajouter des scripts et d'intégrer dans du code Python des appels vers ces outils.

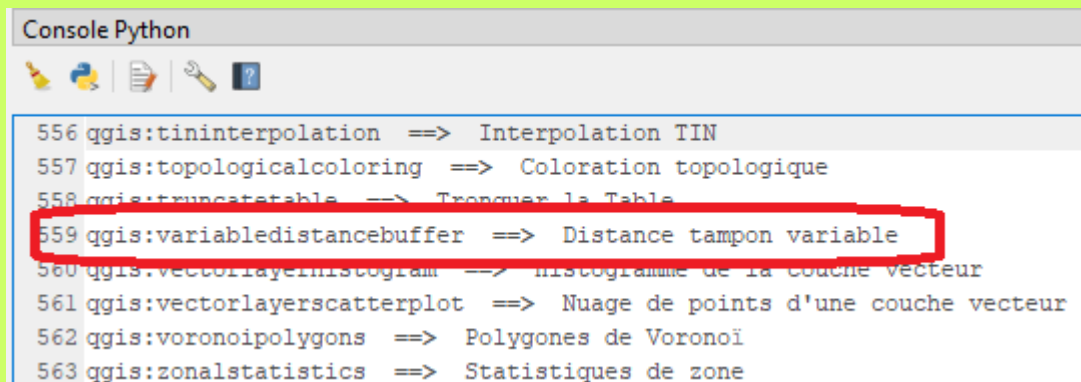
Exemples :

Import du module **processing**

```
import processing
```

```
# liste des algorithmes
```

```
for alg in QgsApplication.processingRegistry().algorithms(): print(alg.id(), ' ==> ', alg.displayName())
```



```
Console Python
556 qgis:tininterpolation ==> Interpolation TIN
557 qgis:topologicalcoloring ==> Coloration topologique
558 qgis:truncatetable ==> Tronquer la Table
559 qgis:variabledistancebuffer ==> Distance tampon variable
560 qgis:vectorlayermistogram ==> histogramme de la couche vecteur
561 qgis:vectorlayerscatterplot ==> Nuage de points d'une couche vecteur
562 qgis:voronoipolygons ==> Polygones de Voronoï
563 qgis:zonalstatistics ==> Statistiques de zone
```

Sources documentaires : http://docs.qgis.org/2.2/fr/docs/user_manual/processing/console.html

QGIS, Aller plus loin ...

Qgis toujours plus ...pour les curieux

Utiliser la boîte à outils « Traitements » :

Les différentes bibliothèques de la boîte à outils sont accessibles depuis le menu « **Traitements** ». Il est possible de construire des modèles, de réaliser des traitements par lot, d'ajouter des scripts et d'intégrer dans du code Python des appels vers ces outils.

Exemples :

affichage des options de fonctions Qgis ...

processing.**algorithmHelp**("qgis:variabledistancebuffer")

```

12 Input parameters
13 -----
14
15 INPUT: Couche en entrée
16
17     Parameter type: QgsProcessingParameterFeatureSource
18
19     Accepted data types:
20         - str: ID de couche
21         - str: nom de couche
22         - str: couche source
23         - QgsProcessingFeatureSourceDefinition
24         - QgsProperty
25         - QgsVectorLayer
26

```

Pourra s'écrire par exemple

```

{ 'DISSOLVE' : False, 'FIELD' : QgsProperty.fromExpression('"densite"'),
  'END_CAP_STYLE' : 0, 'INPUT' :
  'C:/MOC_Q3_FORMATEUR/Donnees/couches/zone_commune_densite.shp',
  'JOIN_STYLE' : 0, 'MITER_LIMIT' : 2, 'OUTPUT' : 'memory:',
  'SEGMENTS' : 5 }

```

Sources documentaires : https://docs.qgis.org/testing/en/docs/user_manual/processing/console.html



QGIS, Aller plus loin ...

Qgis toujours plus ...pour les curieux

Utiliser la boîte à outils « Traitements » :

Les différentes bibliothèques de la boîte à outils sont accessibles depuis le menu « **Traitements** ». Il est possible de construire des modèles, de réaliser des traitements par lot, d'ajouter des scripts et d'intégrer dans du code Python des appels vers ces outils.

Exemples :

exécution

processing.run(name_of_the_algorithm, parameters) ==>Création un objet de type QgsVectorLayer

```
processing.run("qgis:variabledistancebuffer", { 'DISSOLVE' : False, 'FIELD' :  
QgsProperty.fromExpression('"densite"'), 'END_CAP_STYLE' : 0, 'INPUT' :  
'C:/MOC_Q3_FORMATEUR/Donnees/couches/zone_commune_densite.shp', 'JOIN_STYLE' : 0,  
'MITER_LIMIT' : 2, 'OUTPUT' : 'memory:', 'SEGMENTS' : 5 })
```

Sources documentaires : https://docs.qgis.org/testing/en/docs/user_manual/processing/console.html



QGIS, Aller plus loin ...

Qgis toujours plus ...pour les curieux

Utiliser la boîte à outils « Traitements » :

Les différentes librairies de la boîte à outils sont accessibles depuis le menu « **Traitements** ». Il est possible de construire des modèles, de réaliser des traitements par lot, d'ajouter des scripts et d'intégrer dans du code Python des appels vers ces outils.

Exemples :

exécution, création d'une couche mémoire et chargement de la couche dans le CanVas

```
myLayerMemoryDict = processing.run("qgis:variabledistancebuffer", { 'DISSOLVE' : False,  
'FIELD' : QgsProperty.fromExpression('"densite"'), 'END_CAP_STYLE' : 0, 'INPUT' :  
'C:/MOC_Q3_FORMATEUR/Donnees/couches/zone_commune_densite.shp', 'JOIN_STYLE' : 0,  
'MITER_LIMIT' : 2, 'OUTPUT' : 'memory:', 'SEGMENTS' : 5 })
```

```
QgsProject.instance().addMapLayer(myLayerMemoryDict['OUTPUT'])
```

Sources documentaires : https://docs.qgis.org/testing/en/docs/user_manual/processing/console.html



QGIS, Aller plus loin ...

La classe **QgsSettings()** :

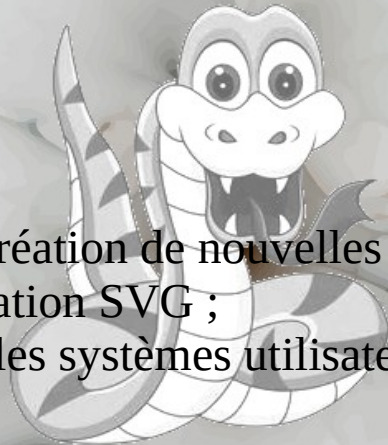
Les paramètres globaux, accessibles depuis le menu « **Préférences/options** » sont normalement liés à l'utilisateur de la machine. QGIS stocke de plus en plus de paramètres globaux.

Exemples :

- Le comportement sur la création de nouvelles couches (définition du SCR) ;
- Les chemins de représentation SVG ;
- Le paramétrage de variables systèmes utilisateur ;
- ...

La librairie **Qt** par le biais de la classe **QgsSettings** fournit le service de stockage et de récupération de ces informations.

La méthode de conservation dépend du système d'exploitation : la **base de registre** (sous **Windows**), le fichier **plist** (sur **Mac OS X**) ou **ini** (sous **Unix**)...



QGIS, Aller plus loin ...

La classe `QgsSettings()` :

Autre **exemple** pour automatiquement définir le bon SCR pour une couche « déduite » (basée sur une couche vecteur chargée dans la session QGIS).

```
#=====
# FONCTIONS CREATION COUCHE AVEC CRS COUCHE ORIGINE
#=====
def ChangeSETTINGS(self, zLayer):
    settings = QgsSettings()
    oldProjectionSetting = settings.value("Projections/defaultBehaviour")
    oldProjectionCRSValue = settings.value("Projections/layerDefaultCrs")
    settings.setValue("Projections/defaultBehaviour", "useGlobal")
    if zLayer == None : zProjEPSG = str(self.iface.mapCanvas().mapRenderer().destinationCrs().authid())
    else : zProjEPSG = str(zLayer.crs().authid())
    settings.setValue("Projections/layerDefaultCrs", zProjEPSG)
    settings.sync()
    return oldProjectionSetting, oldProjectionCRSValue

def DefineLayerProj(self, zLayer, CibleLayer):
    if zLayer == None : CibleLayer.setCrs(self.iface.mapCanvas().mapRenderer().destinationCrs())
    else : CibleLayer.setCrs(zLayer.crs())
    CibleLayer.updateFieldMap()

def RestoreSETTINGS(zProjectionSetting, zProjectionCRSValue):
    settings = QgsSettings()
    if zProjectionSetting != "" : settings.setValue("Projections/defaultBehaviour", zProjectionSetting)
    if zProjectionCRSValue != "" : settings.setValue("Projections/layerDefaultCrs", zProjectionCRSValue)
```



QGIS, Aller plus loin ...

La classe **QgsSettings()** :

Exemple d'appels des trois fonctions de la diapo précédente :

```
zProjectionSetting, zProjectionCRSValue = ChangeSETTINGS(self, zLayer)
AnaLayer = QgsVectorLayer("Point", zLayer.name() + " (SymbProp" + " : " + nLabelFieldName + ")",
"memory")
DefineLayerProj(self, zLayer, AnaLayer)
RestoreSETTINGS(zProjectionSetting, zProjectionCRSValue)
```

Récupération du
paramétrage originel

Affectation du SCR de
la table support

Restauration du
paramétrage originel















QGIS, Aller plus loin ...

Le module « *Shapely* » :

Parmi les normes de l'**Open Geospatial Consortium**, ou **OGC** figure **Simple Features**. Celle-ci définit le modèle géométrique à utiliser pour les entités vectorielles SIG :

- Simple Feature Access - Part 1: Common Architecture:
<http://www.opengeospatial.org/standards/sfa>
- Simple Feature Access - Part 2: SQL Option:
www.opengeospatial.org/standards/sfs



	Point	Ligne	Polygone
			
Intérieur	 dimension 0	 dimension 1	 dimension 2
Limite	n'existe pas (par définition)	 dimension 0	 dimension 1
Extérieur	 dimension 2	 dimension 2	 dimension 2



QGIS, Aller plus loin ...

Le module « *Shapely* » :

En 2007, sous **Python**, **Sean GILLIES** a créé le module **Shapely** pour permettre d'effectuer, en simplifiant, tout ce qu'il est possible de faire avec **PostGIS** au niveau des géométries, sans utiliser ni **SQL** ni **Java**. Il est aussi basé sur la librairie **GEOS**, qui doit être installée auparavant (pour **QGIS**, les distributions **GEOS/Shapely** sont embarquées).

Le module ne traite que les géométries dans un plan cartésien xy, sans projection (SRID). **Sean GILLIES** part du principe qu'il vaut mieux travailler en coordonnées cartésiennes et une fois les démarches effectuées, les projeter ou les re-projeter avec d'autres outils.

Un exemple complet dans le module V !!!

Site officiel :

<http://pypi.python.org/pypi/Shapely/1.2>

<http://pypi.python.org/pypi/descartes>





Formation Python pour QGIS 3



Fin du module

Merci de votre attention !!!!

