DOCUMENT

Centre de Prestations
et d'ingénierie
informatiques

DO Sud-Est

1er février 2011

QGIS Téthys
Notions élémentaires du langage Python
du 1er février 2011

Ressources, territoires, habitats et logement
Énergie et climat
Développement durable
Prévention des risques
Infrastructures, transports et mer

Présent pour l'avenir

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

Ministère de l'Écologie, de l'Énergie, du Développement durable et de la Mer

Ministère de l'Écologie, de l'Énergie,
du Développement durable et de la Mer
en charge des Technologies vertes et des Négociations sur le climat

Historique des versions du document

Version	Date	Commentaire
1.0	01/09/2010	Christophe MASSE - Rédaction initiale
1.1	01/02/2011	Christophe MASSE – Compléments (exemples de code partie 11)

Affaire suivie par

Christophe MASSE – CP2I/DO Sud-Est/ groupe CAS
Tél. : 04 74 27 52 26/ Fax : 04 74 27 52 99
Courriel : christophe.masse@developpement-durable.gouv.fr

Rédacteur

Christophe MASSE - DO Sud-Est / CAS

Relecteur

Prénom NOM - CAS/CMSIG(s)

Référence(s) intranet

Site CMSIG DO Sud-Est *: http://portail-ig.metier.i2/rubrique.php3?id_rubrique=11*

TABLE DES MATIERES

1 - Le langage Python

1.1 - Philosophie et historique du langage

- Au **CWI** (Centrum voor Wiskunde en Informatica) :

À la fin des années 1980, le programmeur Guido van Rossum participait au développement du langage de programmation ABC au Centrum voor Wiskunde en Informatica (CWI) aux Pays-Bas. Il travaillait alors dans l'équipe du système d'exploitation Amoeba dont les appels systèmes étaient difficilement interfaçables avec le bourne shell qui était utilisé comme interface utilisateur. Il estima alors qu'un langage de script inspiré d'ABC pourrait être intéressant comme interpréteur de commandes pour Amoeba

En 1989, profitant d'une semaine de vacances durant les fêtes de Noël, il utilise son Macintosh personnel pour écrire la première version du langage. Fan de la série télévisée des Monty Python, il décide de baptiser ce projet Python. Il s'est principalement inspiré d'ABC, par exemple pour l'indentation comme syntaxe ou les types de haut niveau mais aussi de Modula-3 pour la gestion des exceptions, du langage C et des outils UNIX.

Durant l'année suivante le langage commence à être adopté par l'équipe du projet Amoeba, Guido poursuivant son développement principalement pendant son temps libre. En février 1991, la première version publique, numérotée 0.9.0, est postée sur le forum Usenet alt.sources. La dernière version sortie au CWI fut Python 1.2.

- Au **CNRI** (Corporation for National Research Initiatives) :

En 1995, Van Rossum continua son travail sur Python au CNRI (en) à Reston (en), aux États-Unis, où il sortit plusieurs versions du logiciel.

À partir d'août 1995, l'équipe python travaille au CNRI sur Grail un navigateur web utilisant Tk. Il est l'équivalent pour python du navigateur HotJava, permettant d'exécuter des applets dans un environnement sécurisé. La première version publique, disponible en novembre, est la 0.2. Il a entraîné le développement de modules pour la librairie standard comme rexec, htmllib ou urllib. La version 0.6 sera la dernière de Grail; elle est publiée en avril 1999.

En 1999, le projet Computer Programming for Everybody (CP4E) est lancé avec collaboration entre le CNRI et la DARPA. Il s'agit d'utiliser python comme langage d'enseignement de la programmation. Cette initiative conduira à la création de l'environnement de développement IDLE. Les subventions fournies par la DARPA ne suffisant pas à pérenniser le projet, Guido doit quitter le CNRI. Python 1.6 fut la dernière version sortie au CNRI.

- A **BOpen** :

Après la sortie de Python 1.6, et après que Van Rossum a quitté le CNRI pour travailler avec des développeurs de logiciels commerciaux, le CNRI et la Free Software Foundation collaborèrent pour modifier la licence de Python afin de la rendre compatible avec la GPL. Python 1.6.1 est essentiellement le même que Python 1.6 avec quelques correctifs mineurs et la nouvelle licence compatible GPL.

En 2000, l'équipe principale de développement de Python déménagea à BeOpen.com pour former l'équipe PythonLabs de BeOpen. Python 2.0 fut la seule version sortie à BeOpen.com. Après cette version, Guido Van Rossum et les autres développeurs de PythonLabs rejoignirent Digital Creations.

Andrew M. Kuchling a publié en décembre 1999 un texte nommé python warts qui synthétise les griefs les plus fréquents exprimés à l'encontre du langage. Ce document aura une influence certaine sur les développements futurs du langage.

- **La Python Software Foundation** :

Python 2.1 fut une version dérivée de Python 1.6.1, ainsi que de Python 2.0. Sa licence fut renommée Python Software Foundation License. Tout code, documentation et spécifications ajouté, depuis la sortie de Python 2.1 alpha, est détenu par la Python Software Foundation (PSF), une association sans but lucratif fondée en 2001, modelée d'après l'Apache Software Foundation.

Afin de réparer certains défauts du langage (ex: orientation objet avec deux types de classes), et pour nettoyer la bibliothèque standard de ses éléments obsolètes et redondants, Python a choisi de casser la compatibilité ascendante dans la nouvelle version majeure : Python 3.0, publié en décembre 2008. Cette version a été suivie rapidement par une version 3.1 qui corrige les erreurs de jeunesse de la version 3.0 en la rendant directement obsolète.

1.2 - La communauté Python et les ressources utiles

Van Rossum est le principal auteur de Python, et son rôle de décideur central permanent de Python est reconnu avec humour par le titre de « Dictateur bienveillant à vie » (Benevolent Dictator for Life, BDFL).

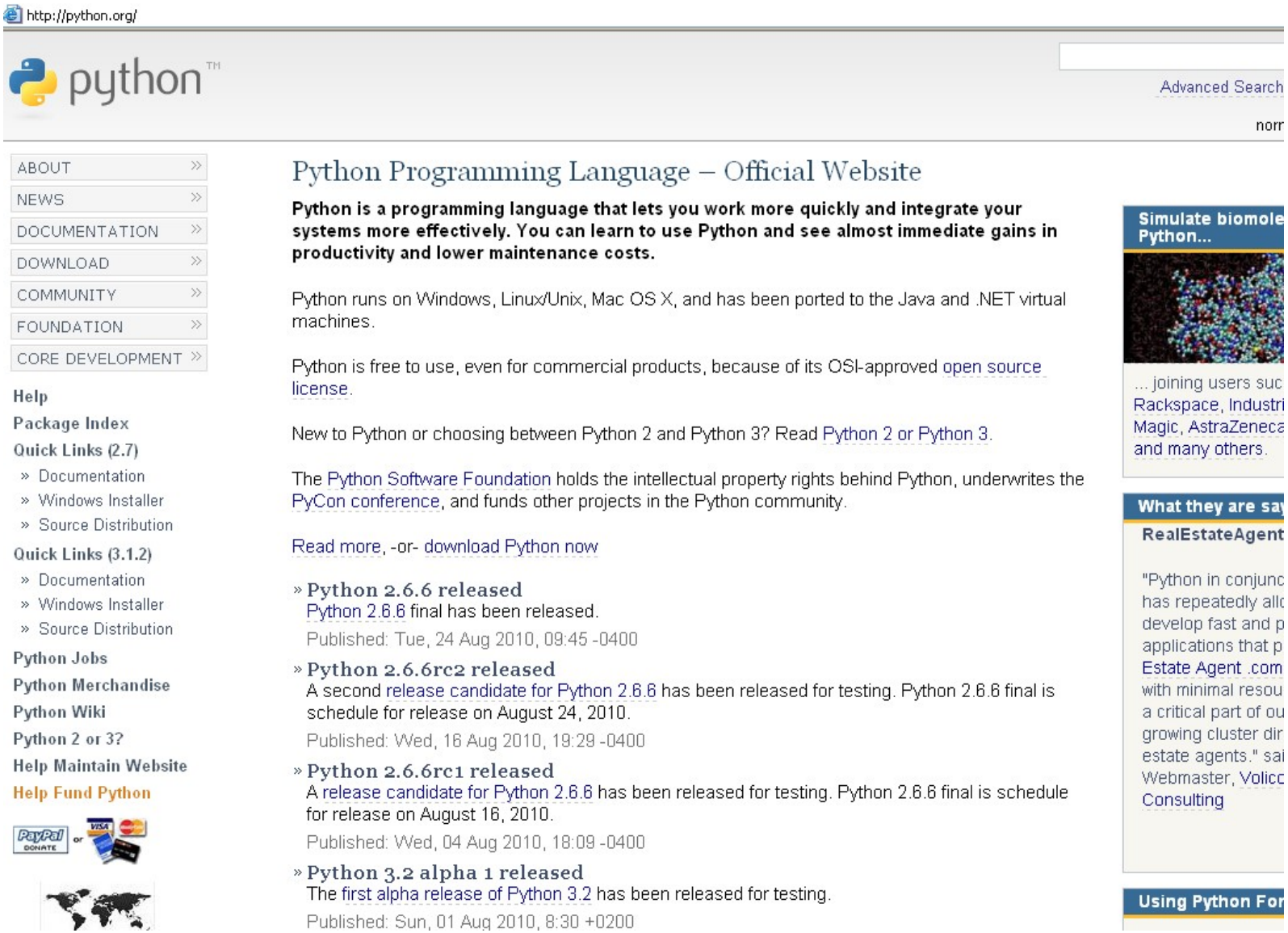
Il est assisté d'une équipe de core developers qui ont un accès en écriture au dépôt de CPython et qui se coordonnent sur la liste de diffusion python-dev. Ils travaillent principalement sur le langage et la bibliothèque de base. Ils reçoivent ponctuellement les contributions d'autres développeurs Python via la plateforme de gestion de bug Roundup, qui a remplacé SourceForge.

Les utilisateurs ou développeurs de bibliothèques tierces utilisent diverses autres ressources. Le principal média généraliste autour de Python est le forum Usenet anglais comp.lang.python.

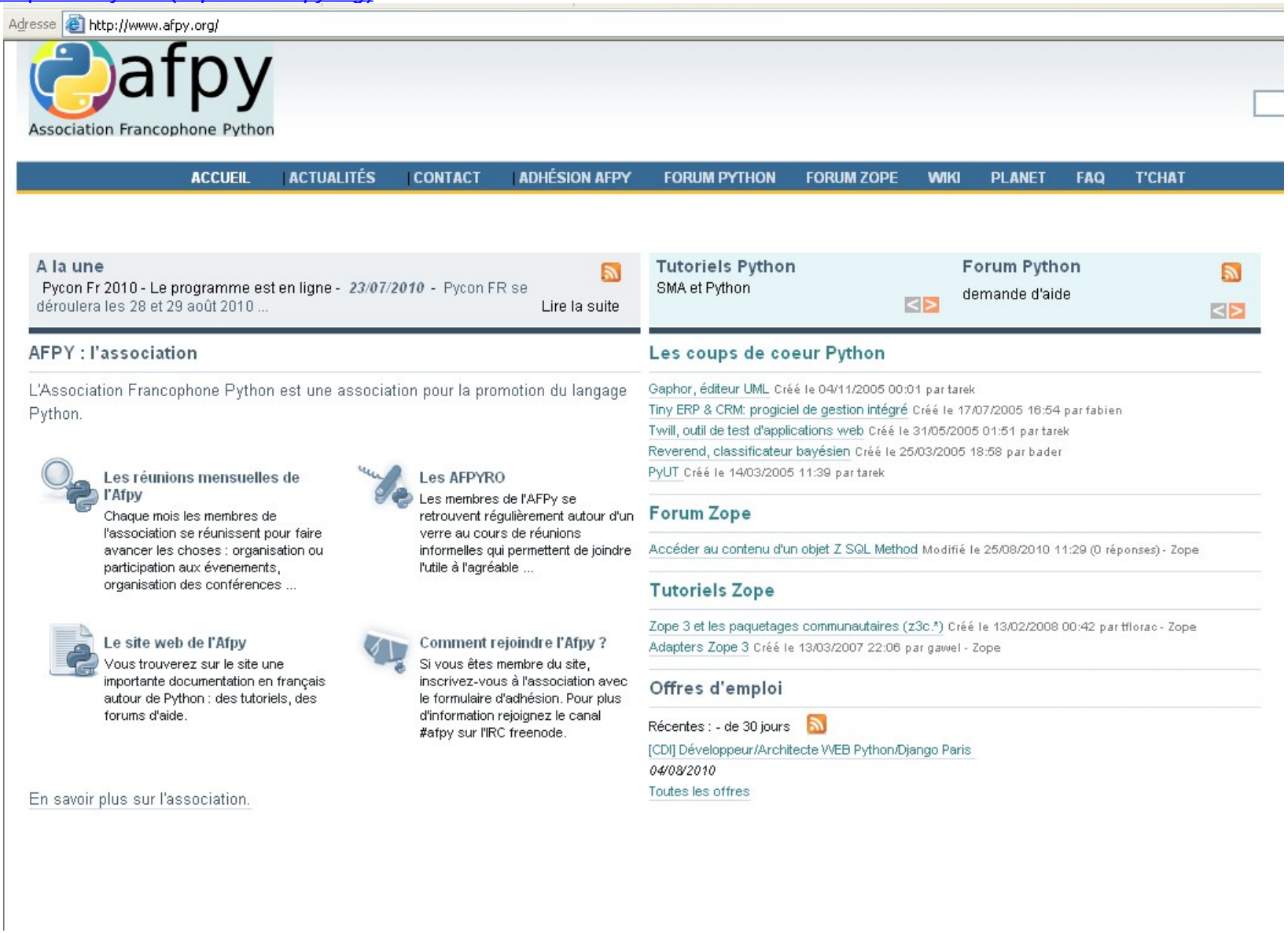
Les allusions aux Monty Python sont assez fréquentes. Les didacticiels consacrés à Python utilisent

souvent les mots spam et eggs comme variable métasyntaxique. Il s'agit d'une référence à l'épisode 25 de la deuxième saison du Monty Python's Flying Circus, ou deux clients tentent de commander un repas à l'aide d'une carte qui contient du jambon en conserve (le spam) dans pratiquement tous les plats. Ce sketch a été aussi pris pour référence pour désigner un email non sollicité.

Le site officiel Python (*http://python.org*)



L'association Francophone Python (*http://www.afpy.org*)



2 - Les principaux outils de développements :

Une des grandes forces du langage Python réside dans le nombre important de bibliothèques logicielles externes disponibles. Une bibliothèque est un ensemble de fonctions. Celles-ci sont regroupées et mises à disposition afin de pouvoir être utilisées sans avoir à les réécrire.

Celles-ci permettent de faire : du calcul numérique, du graphisme, de la programmation internet ou réseau, du formatage de texte, de la génération de documents...

2.1 - Bibliothèques standards

La distribution standard de Python contient un certain nombre de bibliothèques qui ont été considérées comme suffisamment génériques pour intéresser la majorité des utilisateurs.

Leur utilisation est relativement bien expliquée dans la documentation de la distribution.

Les couches de présentation des applications (couche IHM avec wxPython, pyQt, PyKDE Tk, tkinter 3000, pyGTK, pybwidget, Pmw, TIX)

les couches controller des serveurs d'application Web (analyse HTML -htmllib, xmllib, urlParse, mimetools- Serveur d'application : Zope - Django, Turbogears, CherryPy, Plone, GGI)

les couches Modele d'accès aux données (MySQL -MySQLdb- , Oracle -dcoracle-, MS SQL Server, PostgreSQL -psycopg-, FireBird -kinterbasdb- , SybODBC, GadFly, PySqlite, Zodb- BDD objet -)

la couche de persistance XML (4Suite, PySimpleXML, XmlSerializer, Jaxml) ou spécifique à Python (Cpickle, Shelve)

les couches d'accès au middleware ICE, COM/CORBA/.NET (win32, OmniORB, Ironpython) : programmation orientée composant (pont vers des bibliothèques Fortran, C et C++)

les couches de communication standalone (port série : pySerial, port parallèle : pyParallel) , réseau (Twisted, urllib2, HTMLparser, ftplib], socket, poplib, rfc822, mailbox, mhlib, imaplib, smtplib, telnet, etc.)

les couches de frameWork bas niveau (ajout de capacité de script. exemple : Boost.Python)
Les couches multimédia : traitement d'image (PIL)
Les couches utilitaires :

- de gestion de l'heure (datetime, time)
- de compression (gzip)
- de codage/décodage de données binaires (hashlib -md5, sha- , base64, binHex, binascii)
- de structure de données (decimal, deque, array, dico, list, queue, heapq)
- de parallélisme (thread)
- d'expressions régulières (re)
- de différences (difflib)
- d'acces au dll ou.so (ctype)
- de manipulation de chaînes (string, str, StringIO)
- de parseur (standard - shlex, ConfigParser, email, parser, cmd - ou Tierce- pyBison, ples, pyparsing, ply, PyGgy, Yapps, pyLR)

- de calcul (math, numarrray - tableaux multidimensionnaires - , cmath, random)
- de log (logging)

Le déploiement se fait soit en utilisant des modules d'installation standardisés (distutils), soit en générant un exécutable qui ne nécessite plus l'existence de l'interpréteur sur les machines cibles (Windows : py2exe, Cx_freeze; Unix : freeze)

2.2 - Bibliothèques alternatives

- Un site internet recense la liste des bibliothèques utilisables avec ce langage: http://www.vex.net/parnassus
- Les bibliothèques les plus populaires et les plus utilisées (XML, interfaces graphiques...) bénéficient de pages dédiées sur le site principal du langage : http://cheeseshop.python.org/pypi?%3Aaction=index

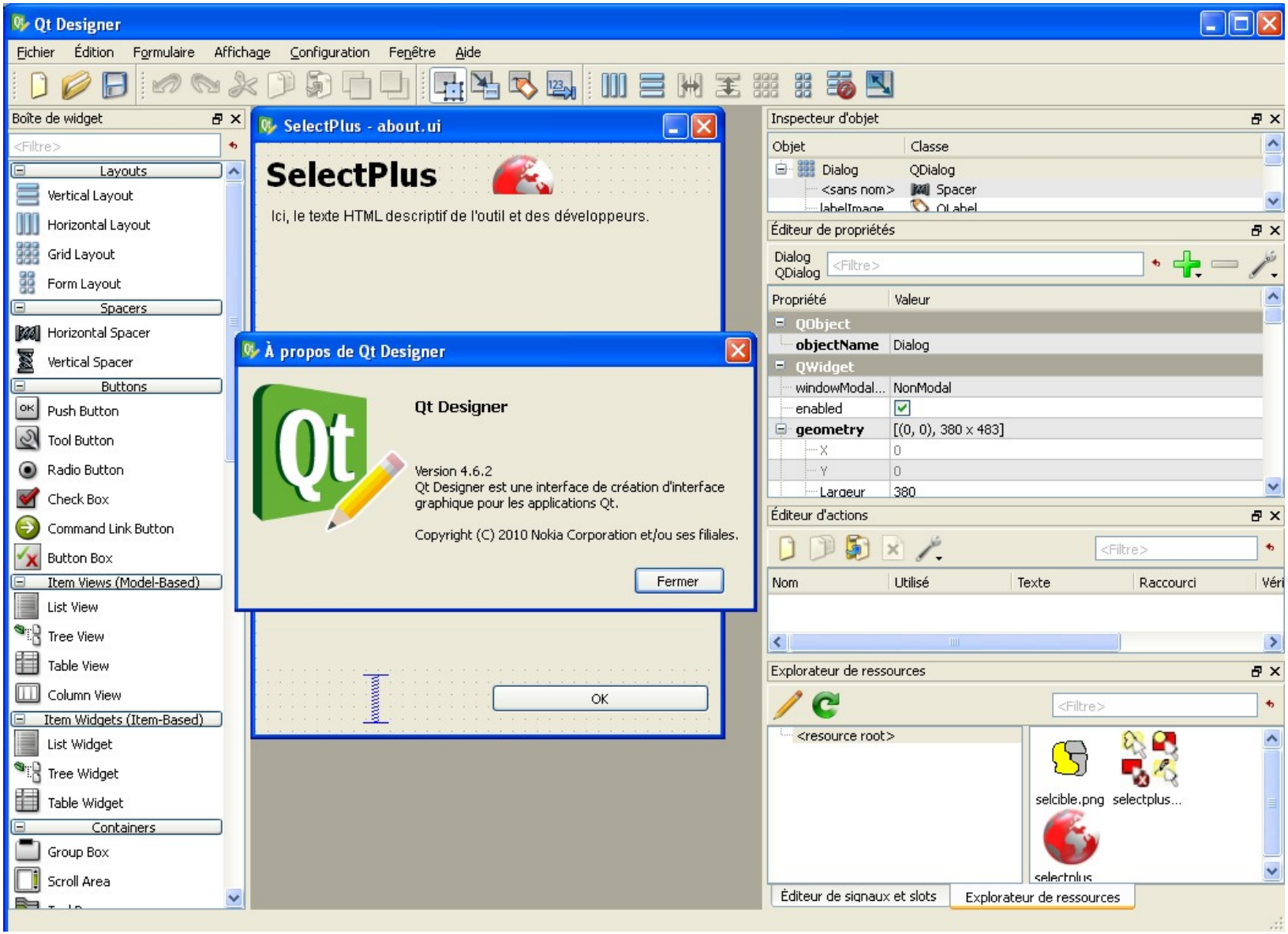
2.3 - Liste de Bibliothèques

- CWM : Modules de parseur pour le web semantique :http://infomesh.net/2001/cwm/
- epydoc : Utilisé pour générer la documentation : http://epydoc.sourceforge.net/
- SAGE : Logiciel d'algèbre et de géometrie (alternative à MAthematica, Maple ...) : géométrie, théorie des nombres, cryptographie, calcul numérique... : http://modular.math.washington.edu/sage/
- pyCLIPS : Module pour scripter le système expert CLIPS : http://pyclips.sourceforge.net/
- Pychinko: Implémentation de l'algorithme de Rete (pour le chaînage avant)
- pydot : Module pour scripter le générateur de graphique Graphviz : http://dkbza.org/pydot.html
- Pygame : Module de création de jeu 2D : http://www.pygame.org/news.html
- PySFML : Module de création de jeu 2D : http://www.sfml-dev.org/tutorials/start-python-fr.php
- Pyglet : Module de création de jeu 2D utilisant l'openGL : http://www.pyglet.org/
- Cocos2d : Frameworks multiplateformes pour construire des jeux 2d, demos ou des applications interactives graphiques en openGL (nécessite Pyglet) : http://cocos2d.org/
- Karrigell, CherryPy , TurboGears ,Django , Web2py, Pylons : Framework de développement web.
- Soya : Moteur 3D :http://home.gna.org/oomadness/en/soya3d/index.html
- Panda 3D : Moteur 3D : http://www.panda3d.org/
- PyOgre : Moteur 3D : http://www.ogre3d.org/
- Pmw, Pybwidget, Tkinter 3000, PyGTK : Interface graphique.
- pyinstaller : Création d'exécutable pour toute plateforme : http://pyinstaller.python-hosting.com/
- Py2exe : Créer un exécutable Windows pour vos scripts : http://www.py2exe.org/
- MySQLdb, Gadfly, Psycopg, Kinterbasdb, Buzhug : Base de données.
- Matplotlib : Bibliothèque de dessin de courbe 2D (très riche) : http://matplotlib.sourceforge.net/
- gnuplot-py : Bibliothèque pour s'interfacer avec gnuplot : http://gnuplot-py.sourceforge.net/
- PyNGL : Bibliothèque pour créer des graphes 2D : http://www.pyngl.ucar.edu/index.shtml
- scipy et numpy : Framework pour le calcul scientifique : Interpolation, intégration (ODE integrators), algèbre linéaire (LAPACK), Interpolation, systèmes dynamiques (PyDSTool) : http://www.scipy.org/
- PyIMSL Studio : Framework pour le calcul scientifique s'appuyant sur les bibliothèques mathématiques et statistiques IMSL :

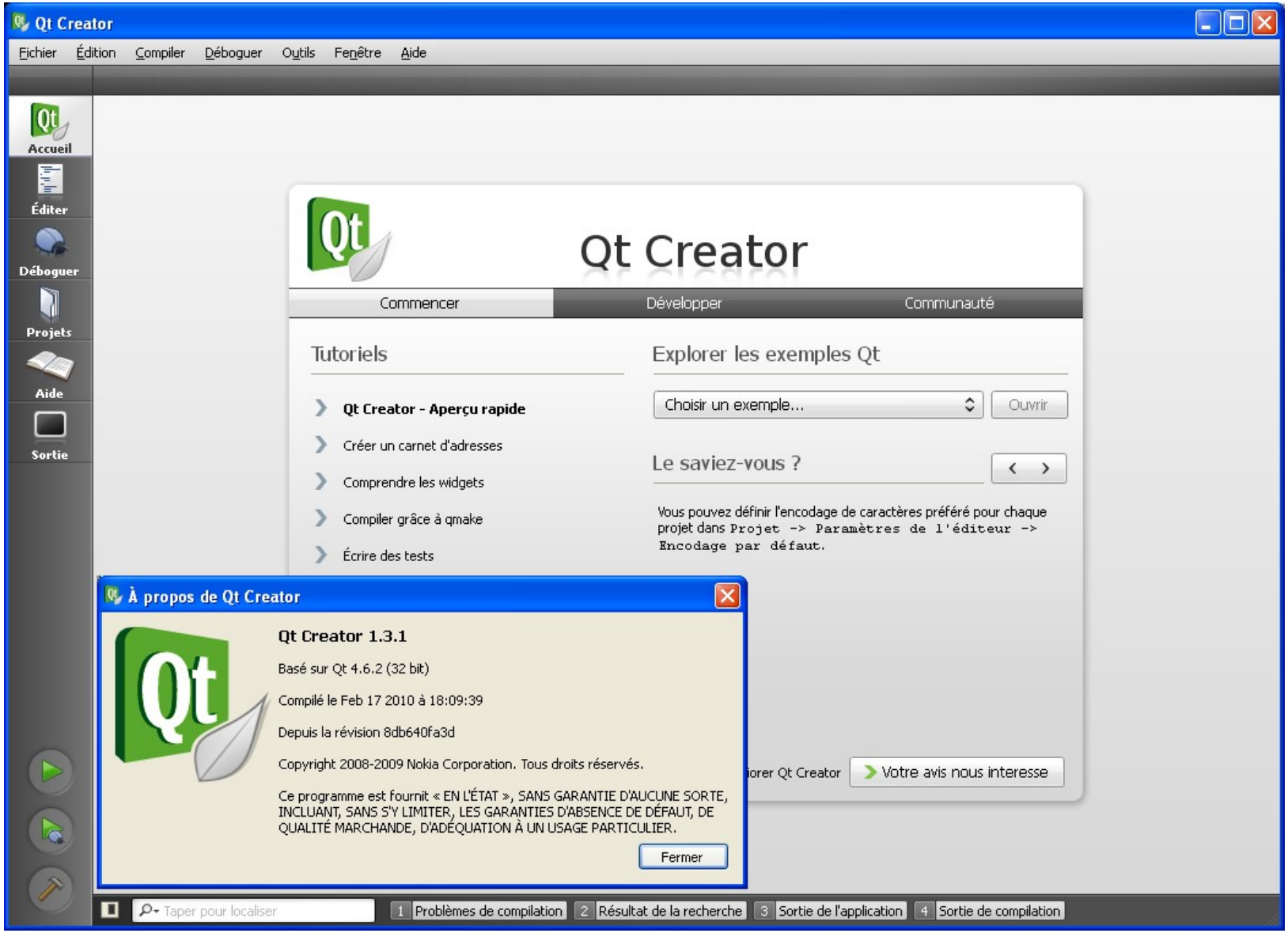
http://sites.google.com/site/visualnumerics/produits/pyimslstudio
- PIL : Manipulation et traitement d'image : http://www.pythonware.com/products/pil/
- SVGdraw: Création d'image au format SVG (Scalable Vector Graphics) : http://www2.sfk.nl/svg
- pygsl : Interface vers GNU scientific library (gsl): vecteur, matrice, transformation de fourrier, recuit simulé, algèbre linéaire... : http://pygsl.sourceforge.net/
- CGAL: CGAL-Python bindings pour la CGAL library (Computational Geometry Algorithms Library) : http://cgal-python.gforge.inria.fr/
- GMPY : General Multiprecision PYthon : Interface vers la bibliothèque de précision arithmétique GNU GMP : http://gmpy.sourceforge.net/
- pyrorobotics : Environnement pour l'étude de la robotique et l'intelligence artificielle. Réseaux de neurones : http://www.pyrorobotics.org/
- pybel : Interface pour la bibliothèque Open source de CHIMIE Open Babel.
- FANN : Fast Artificial Neural Network Library : binding Python pour FANN : http://leenissen.dk/fann/index.php
- Maximum Entropy Modeling Toolkit : Framework qui met en oeuvre le principe de l'entropie maximum : http://homepages.inf.ed.ac.uk/s0450736/maxent_toolkit.html
- Orange : Technique d'exploration de données, data mining : http://www.ailab.si/orange
- MayaVi2 : Visualisation des données scientifiques en 2D/3D : https://svn.enthought.com/enthought/wiki/MayaVi
- pySerial : Manipulation des ports séries, non intégrés par défaut, bien que très souvent utilisés dans le monde industriel : http://pyserial.sourceforge.net/
- pyParallel : Accès aux ports parallèles : http://pyserial.sourceforge.net/pyparallel.html
- pyMPI : Calcul parallèle : http://pympi.sourceforge.net/index.html
- PyPar : Calcul parallèle : http://datamining.anu.edu.au/~ole/pypar/
- PyVISA : Contrôle des ports GPIB, RS232, and USB : http://pyvisa.sourceforge.net/
- PyUSB : Manipulation du port USB : http://pyusb.berlios.de/
- pyro : Middleware Python Remote Objects : http://pyro.sourceforge.net/
- pyX : Python graphics package - Analyse de donnée : http://pyx.sourceforge.net/
- simPy : Simulation de systèmes dynamiques à file d'attente : http://simpy.sourceforge.net/
- Twisted : Pluggable, asynchronous protocols (TCP, UDP, SSL/TLS, multicast, Unix sockets, HTTP, NNTP, IMAP, SSH, IRC, FTP) : http://twistedmatrix.com/trac/
- SCons : Alternative puissante à make (next-generation build tool) : http://www.scons.org/
- VPython : Simulation 3D basée sur OpenGl : http://www.vpython.org/
- directpython: Binding vers Direct X : http://directpython.sourceforge.net/
- pymedia : Module for wav, mp3, ogg, avi, divx, dvd, cdda etc. files manipulations : http://pymedia.org/
- wxPython : Bibliothèque d'accès à un toolkit très puissant (en particulier pour les interfaces graphiques) : http://www.wxpython.org/
- PyML : Python machine learning package : Framework pour l'apprentissage par les machines (data mining ...) : http://pyml.sourceforge.net/
- Zope : Serveur d'application web orienté objet et base de données Objet : http://www.zope.org/

La société **NOKIA** met toujours à disposition l'environnement « **Qt** » pour la réalisation de solutions avec Python. « Qt » est disponible en version « light » (« PyQt ») ou en version lourde (« SDK »). Ci-dessous, des captures d'écrans des deux environnements très proches, mais avec un panel fonctionnel moindre dans la version « light ».

PyQT 4.6.2 :


Qt Designer
Fichier Édition Formulaire Affichage Configuration Fenêtre Aide
Boîte de widget
<Filtre>
Layouts
Vertical Layout
Horizontal Layout
Grid Layout
Form Layout
Spacers
Horizontal Spacer
Vertical Spacer
Buttons
Push Button
Tool Button
Radio Button
Check Box
Command Link Button
Button Box
Item Views (Model-Based)
List View
Tree View
Table View
Column View
Item Widgets (Item-Based)
List Widget
Tree Widget
Table Widget
Containers
Group Box
Scroll Area
SelectPlus - about.ui
SelectPlus
Ici, le texte HTML descriptif de l'outil et des développeurs.
À propos de Qt Designer
Qt Designer
Version 4.6.2
Qt Designer est une interface de création d'interface graphique pour les applications Qt.
Copyright (C) 2010 Nokia Corporation et/ou ses filiales.
Fermer
OK
Inspecteur d'objet
Objet
Classe
Dialog
QDialog
<sans nom>
Spacer
Éditeur de propriétés
Dialog
QDialog
<Filtre>
Propriété
Valeur
QObject
objectName
Dialog
QWidget
windowModal...
NonModal
enabled
geometry
[(0, 0), 380 x 483]
X
Y
Largeur
380
Éditeur d'actions
<Filtre>
Nom
Utilisé
Texte
Raccourci
Explorateur de ressources
<Filtre>
<resource root>
selcible.png
selectplus...
Éditeur de signaux et slots
Explorateur de ressources

SDK 1.3.1 :


Qt Creator
Fichier Édition Compiler Déboguer Outils Fenêtre Aide
Accueil
Éditer
Déboguer
Projets
Aide
Sortie
Qt Creator
Commencer
Développer
Communauté
Tutoriels
Qt Creator - Aperçu rapide
Créer un carnet d'adresses
Comprendre les widgets
Compiler grâce à qmake
Écrire des tests
Explorer les exemples Qt
Choisir un exemple...
Ouvrir
Le saviez-vous ?
Vous pouvez définir l'encodage de caractères préféré pour chaque projet dans Projet -> Paramètres de l'éditeur -> Encodage par défaut.
Votre avis nous interesse
À propos de Qt Creator
Qt Creator 1.3.1
Basé sur Qt 4.6.2 (32 bit)
Compilé le Feb 17 2010 à 18:09:39
Depuis la révision 8db640fa3d
Copyright 2008-2009 Nokia Corporation. Tous droits réservés.
Ce programme est fournit « EN L'ÉTAT », SANS GARANTIE D'AUCUNE SORTE, INCLUANT, SANS S'Y LIMITER, LES GARANTIES D'ABSENCE DE DÉFAUT, DE QUALITÉ MARCHANDE, D'ADÉQUATION À UN USAGE PARTICULIER.
Fermer
Taper pour localiser
1 Problèmes de compilation
2 Résultat de la recherche
3 Sortie de l'application
4 Sortie de compilation

3 - Les caractéristiques du langage : rôle de l'indentation, l'encodage, les commentaires, l'import de librairies...

Sous Python, l'indentation est très importante. Elle structure les instructions du programme.

Exemple de code :

```
    lstWMS = ""
    for i in range(len(nParamWMS)):
        tempo = ExtractValWMS(nWMS,str(nParamWMS[i]))
        if tempo !="":
            lstWMS = lstWMS + tempo + "|"
    return lstWMS
```

De même, la casse est à respecter scrupuleusement : ainsi dans le code ci-dessus, si « **extractValWMS** » est en lieu et place de « **ExtractValWMS** », l'interpréteur retournera une erreur (variable ou fonction non définie).

Ce respect de la casse vaut pour tous les nommages, qu'il s'agisse de fonctions, de varaibles ou de résultats de tests (logiques).

Pour les commentaires, sous Python comme sous la plupart des langages, il est possible de mettre une ligne en commentaire ou un bloc de lignes en commentaires.

Exemple de code :

Pour une ligne, le caractère « **#** » est placé au début de la ligne.

```
# (c) Christophe MASSE 2010
```

Pour un bloc, les trois caractères « **"""** » doivent marquer le début et la fin du bloc :

```
"""
/*****************************************************************************
Les commentaires sont placés ici.
 *****************************************************************************/
"""
```

Pour l'encodage, par défaut les scripts Python sont encodés en UTF-8. Cet encodage peut généré des erreurs sur des retours de variables (ex : nom de fichier avec des caractères accentués ou spéciaux). Il est possible alors de spéciifier l'encodage employé.

Exemple de code :

```
import sys
reload(sys)
    sys.setdefaultencoding( "iso-8859-1" )
```

La précision de l'encodage du module en entête peut se révéler dans certains cas insuffisantes

```
# -*- coding: utf-8 -*-
```

```
# -*- coding: iso-8859-1 -*-
```

Pour utiliser des fonctions des librairies disponibles sous Python, il faut passer par la commande « import » ces instructions.

Exemple de code :

```
import sys
from math import pi,sin,cos,sqrt
from PyQt4 import QtCore, QtGui
from PyQt4.QtCore import *
from PyQt4.QtGui import *
```

Notez dans les exemples ci-dessus, que ces différentes formes d'appels n'ont pas les mêmes portées.

4 - Les types de variables et leur utilisation

Les types de base en Python sont relativement complets et puissants, il y a entre autres :

4.1 - Les objets numériques

- int est un entier illimité. Avant la version 3.0, ce type était dénommé long, et le type int correspondait à un entier 32 ou 64 bits. Néanmoins, une conversion automatique évitait tout débordement.
- float est un flottant équivalent au type double du C, soit un nombre entre -1,7×10308 et 1,7×10308
- complex est une approximation d'un nombre complexe (typiquement deux floats)

4.2 - Les objets "itérables"

- Les objets **tuple** (ou n-uplet) sont des listes non mutables d'objets hétérogènes.

Exemple de code avec un ***tuple*** :

```
        nParamWMS=("GetMap=\"","<SRS>","<Layer>","wms_version=\"","minx=\"","miny=\"","maxx=\"","maxy=\"","\"
format=\"")
    lstWMS = ""
    for i in range(len(nParamWMS)):
        tempo = ExtractValWMS(nWMS,str(nParamWMS[i]))
        if tempo !="":
            lstWMS = lstWMS + tempo + "|"
```

- Les objets **list** sont des tableaux dynamiques (ils étendent automatiquement leur taille lorsque nécessaire) et acceptent des types de données hétérogènes.

Exemple de code :

```
nParamWMS = QstringList() <<
("GetMap=\"","<SRS>","<Layer>","wms_version=\"","minx=\"","miny=\"","maxx=\"","maxy=\"","\" format=\"")
```

Exemple de code :

```
workDir = QDir( inDir )
-    workDir.setFilter( QDir.Files | QDir.NoSymLinks | QDir.NoDotAndDotDot )
-    nameFilter = QStringList() << "*.shp" << "*.SHP"
-    workDir.setNameFilters( nameFilter )
```

Exemple de code :

```
themeDir = QDir(mAppDir + "/share/qgis/themes")
   themeDir.setFilter(QDir.Dirs);
   dirs = QStringList() << themeDir.entryList("*")
   for i in range(dirs.count()) :
       if(dirs[i] != "."  and dirs[i] != "..") :
          #action1
       else :
          #action2
```

L'opérateur « **<<** » peut être employé pour alimenter la liste.

- Les objets **dict** sont des tableaux associatifs (ou dictionnaires) permettant d'associer un objet (une clef) à une valeur :

Exemple de code :

```
#initialisation d'un dictionnaire
tlayer={}

#ajout d'une entrée (clef / valeur)
tlayer[tkey] = tvalue

#test de la présence d'une clef dans le dictionnaire
if tlayer.has_key(tkey):
                ...

#obtenir toutes les clefs
print tlayer.keys()
#obtenir toutes les valeurs
print tlayer.values()
#obtenir les couples
print tlayer.items()

#il est possible de créer des listes de dictionnaires pour accéder aux « valeurs » partagées par des clefs
identiques
alllayers = [tlayer, ulayer]
for i in alllayers:
    print i['parcellaire']

#retourne :
native (pour tlayer)
C:\Mes documents\parcelle.tab (pour ulayer)
# ces valeurs sont données à titre d'exemple
```

Avec ce dernier exemple, on note que les dictionnaires permettent de se rapprocher d'un comportement de type base de données. On pourrait croire qu'une entrée n'attend qu'une valeur unique (type alpha-numérique ou numérique), il n'en est rien. Les listes, dictionnaires, tuples sont des objets qui peuvent contenir des collections d'autres objets. On peut donc construire des listes qui contiennent des dictionnaires, des tuples ou d'autres listes, mais aussi des dictionnaires contenant des tuples, des listes, etc.

- Les objets **set** sont des ensembles non ordonnés d'objets.
- Les objets (**frozenset**) forment une variante non mutable des set.
- Les objets str sont des chaînes de caractères. À partir de la version 3.0, les caractères sont encodés Unicode sur 16 ou 32 bits ; les chaines d'octets ASCII sont des objets bytes. Dans les versions précédentes, ces objets étaient respectivement de type unicode et str.

Les objets itérables sont parcourus à l'aide d'une boucle for de la manière suivante :

for element in objet_iterable:
*** traiter(element)***

Pour une chaîne de caractère, l'itération a lieu caractère par caractère. Un caractère est une chaîne de longueur 1.

Il est possible de dériver les classes des types de base pour créer ses propres types. On peut également fabriquer ses propres types d'objets itérables sans hériter des itérables de base en implémentant le protocole d'itération du langage.

La fonction « **type()** » permet de connaître le type d'une variable.

Exemple de code :

```
>>> a=3
>>> type(a)
<type 'int'>
```

5 - Contrôle du flux des instructions

5.1 - l'instruction if

Exemple de code :

```
geomc = featc.geometry()
        if main.nOP == 0:
          if geomc.within(g):
            selectedSetc.append(featc.id())
        elif main.nOP == 1:
          if geomc.overlaps(g):
            selectedSetc.append(featc.id())
        elif main.nOP == 2:
          if geomc.contains(g):
            selectedSetc.append(featc.id())
        elif main.nOP == 3:
          if geomc.crosses(g):
            selectedSetc.append(featc.id())
        elif main.nOP == 4:
          if geomc.disjoint(g):
            selectedSetc.append(featc.id())
        elif main.nOP == 5:
          if geomc.equals(g):
            selectedSetc.append(featc.id())
        elif main.nOP == 6:
          if geomc.intersects(g):
            selectedSetc.append(featc.id())
        elif main.nOP == 7:
          if geomc.touches(g):
            selectedSetc.append(featc.id())
```

Notez que le « **Else If** » en Python se traduit par « **elif** ». Inutile de placer des « end if », l'indentation indique la fin d'un bloc d'instruction « if » à l'interpréteur. Sur un test simple, vous pouvez aussi utiliser l'instruction « **else** ».

5.2 - Les boucles for

Exemple de code :

```
    zIndex = 0
    nLayers = self.iface.mapCanvas().layerCount()
    for i in range(nLayers):
        cLayer = self.iface.mapCanvas().layer(i)
        if str(cLayer.getLayerID())== str(vLayer.getLayerID()):
          break
```

```
        zIndex = zIndex + 1
    return zIndex
```

L'instruction « **break** » permet de sortir de la boucle « **for** » si la condition (test « **if** ») est vérifiée.

5.3 - Les boucles while

Exemple de code :

```
    while posd > posf:
        posf = nStr.find("<",posd)
        if posf > posd:
            if zLayersWMS == "":
                zLayersWMS = nStr[posd:posf]
            else:
                zLayersWMS = zLayersWMS + "," + nStr[posd:posf]
        posd = nStr.find("<Name>", posf)+len("<Name>")
```

6 - Les fonctions

6.1 - Définition et appel d'une fonction

Dans un module Python, la définition d'une fonction se décline de la manière suivante :

Exemple de code :

```
def CorrigePath(nPath):
    nPath = str(nPath)
    a = len(nPath)
    subC = "/"
    b = nPath.rfind(subC, 0, a)
    if a != b :
       return (nPath + "/")
    else:
       return nPath
```

Une fonction retourne en règle générale un résultat (passé par le « **return** »).
Notez qu'en Python, il est inutile de typer la ou les variables passées à la fonction.

Exemple de code :

```
#pour la fonction CorrigePath ci-dessus
nDir = CorrigePath(os.path.dirname(nDir))
#autre exemple d'appel d'une fonction chaîne de caractères sur une variable chaîne de caractères
nstring = nstring.upper()
#pour la fonction « upper() », cf. paragraphe ci-après.
```

Dans l'exemple ci-dessus, on note les différentes formes d'appel. Exemple
« **CorrigePath(os.path.dirname(nDir))** »

La variable passée à la fonction « **CorrigePath** » est le résultat d'un appel à une fonction de haut niveau (« **dirname** »).

6.2 - Les principales fonctions sur chaîne de caractères

_apply = apply(...)
apply(object, args[, kwargs]) -> value

Call a callable object with positional arguments taken from the tuple args,
and keyword arguments taken from the optional dictionary kwargs.
Note that classes are callable, as are instances with a __call__() method.

_float = float(...)
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

_**int** = int(...)
int(x[, base]) -> integer

Convert a string or number to an integer, if possible. A floating point argument will be truncated towards zero (this does not include a string representation of a floating point number!) When converting a string, use the optional base. It is an error to supply a base when converting a non-string.

_**long** = long(...)
long(x) -> long integer
long(x, base) -> long integer

Convert a string or number to a long integer, if possible. A floating point argument will be truncated towards zero (this does not include a string representation of a floating point number!) When converting a string, use the given base. It is an error to supply a base when converting a non-string.

atof(s)
atof(s) -> float

Return the floating point number represented by the string s.

atoi(s, base=10)
atoi(s [,base]) -> int

Return the integer represented by the string s in the given base, which defaults to 10. The string s must consist of one or more digits, possibly preceded by a sign. If base is 0, it is chosen from the leading characters of s, 0 for octal, 0x or 0X for hexadecimal. If base is 16, a preceding 0x or 0X is accepted.

atol(s, base=10)
atol(s [,base]) -> long

Return the long integer represented by the string s in the given base, which defaults to 10. The string s must consist of one or more digits, possibly preceded by a sign. If base is 0, it is chosen from the leading characters of s, 0 for octal, 0x or 0X for hexadecimal. If base is 16, a preceding 0x or 0X is accepted. A trailing L or l is not accepted, unless base is 0.

capitalize(s)

capitalize(s) -> string

Return a copy of the string s with only its first character capitalized.

capwords(s, sep=None)
capwords(s, [sep]) -> string

Split the argument into words using split, capitalize each word using capitalize, and join the capitalized words using join. Note that this replaces runs of whitespace characters by a single space.

center(s, width)
center(s, width) -> string

Return a center version of s, in a field of the specified width. padded with spaces as needed. The string is never truncated.

count(s, *args)
count(s, sub[, start[,end]]) -> int

Return the number of occurrences of substring sub in string s[start:end]. Optional arguments start and end are interpreted as in slice notation.

expandtabs(s, tabsize=8)
expandtabs(s [,tabsize]) -> string

Return a copy of the string s with all tab characters replaced by the appropriate number of spaces, depending on the current column, and the tabsize (default 8).

find(s, *args)
find(s, sub [,start [,end]]) -> in

Return the lowest index in s where substring sub is found, such that sub is contained within s[start,end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

index(s, *args)
index(s, sub [,start [,end]]) -> int

Like find but raises ValueError when the substring is not found.

join(words, sep=' ')

join(list [,sep]) -> string

Return a string composed of the words in list, with intervening occurences of sep. The default separator is a single space.

(joinfields and join are synonymous)

joinfields = join(words, sep=' ')
join(list [,sep]) -> string

Return a string composed of the words in list, with intervening occurences of sep. The default separator is a single space.

(joinfields and join are synonymous)

ljust(s, width)
ljust(s, width) -> string

Return a left-justified version of s, in a field of the specified width, padded with spaces as needed. The string is never truncated.

lower(s)
lower(s) -> string

Return a copy of the string s converted to lowercase.

lstrip(s)
lstrip(s) -> string

Return a copy of the string s with leading whitespace removed.

maketrans(...)
maketrans(frm, to) -> string

Return a translation table (a string of 256 bytes long) suitable for use in string.translate. The strings frm and to must be of the same length.

replace(s, old, new, maxsplit=-1)
replace (str, old, new[, maxsplit]) -> string

Return a copy of string str with all occurrences of substring old replaced by new. If the optional argument maxsplit is given, only the first maxsplit occurrences are replaced.

rfind(s, *args)

rfind(s, sub [,start [,end]]) -> int

Return the highest index in s where substring sub is found, such that sub is contained within s[start,end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex(s, *args)
rindex(s, sub [,start [,end]]) -> int

Like rfind but raises ValueError when the substring is not found.

rjust(s, width)
rjust(s, width) -> string

Return a right-justified version of s, in a field of the specified width, padded with spaces as needed. The string is never truncated.

rstrip(s)
rstrip(s) -> string

Return a copy of the string s with trailing whitespace removed.

split(s, sep=None, maxsplit=-1)
split(s [,sep [,maxsplit]]) -> list of strings

Return a list of the words in the string s, using sep as the delimiter string. If maxsplit is given, splits into at most maxsplit words. If sep is not specified, any whitespace string is a separator.

(split and splitfields are synonymous)

splitfields = split(s, sep=None, maxsplit=-1)
split(s [,sep [,maxsplit]]) -> list of strings

Return a list of the words in the string s, using sep as the delimiter string. If maxsplit is given, splits into at most maxsplit words. If sep is not specified, any whitespace string is a separator.

(split and splitfields are synonymous)

strip(s)
strip(s) -> string

Return a copy of the string s with leading and trailing whitespace removed.

swapcase(s)
swapcase(s) -> string

Return a copy of the string s with upper case characters converted to lowercase and vice versa.

translate(s, table, deletions='')
translate(s,table [,deletechars]) -> string

Return a copy of the string s, where all characters occurring in the optional argument deletechars are removed, and the remaining characters have been mapped through the given translation table, which must be a string of length 256.

upper(s)
upper(s) -> string

Return a copy of the string s converted to uppercase.

zfill(x, width)
zfill(x, width) -> string

Pad a numeric string x with zeros on the left, to fill a field of the specified width. The string x is never truncated.

6.3 - Les principales fonctions mathématiques

- x **+** y somme de *x et y*
- x **-** y difference entre *x et y*
- x ***** y produit de *x et y*
- x **/** y division de *x par y*
- x **%** y reste de la division x / y
- **-**x passage en négatif de *x*
- **+**x *x* inchangé
- **abs**(x) valeur absolue or magnitude of *x*
- **int**(x) *x* converti en type integer
- **long**(x) *x* converti en type long integer
- **float**(x) *x* converti en type to float
- **complex**(re,im) definir un nombre complexe avec une composante réelle *re*, une composante imaginaire *im*. *im* par défaut à zéro.
- c.**conjugate**() conjugaison du nombre complexe *c*.

- **pow**(x, y) *x* puissance *y*
- x ****** y *x* puissance *y*

Cette liste n'est pas exhaustive. On peut aussi citer :
math.floor() pour un arrondi inférieur.
math.ceil() pour un arrondi supérieur.
round(x[, n]) pour les arrondis en précisant n (à défaut valeur zéro).

6.4 - Les traitements logiques

Les opérateurs « **and** », « **or** » et « **not** » sont disponibles. Attention à la casse !
Idem pour les résultats en test : « **False** » et pas « false », « **True** » et pas « true ».

Exemple de code :

```
if debFields == True and CountFields > 0:
        CountFields = CountFields - 1
        if astring.startswith(nFieldName):
           zIndex = i
           break
        i = i + 1
```

Pour tester l'égalité de la valeur, comme en Java, l'opérateur est « **==** ». L'opérateur « = » seul assigne une valeur.
Pour le reste, les opérateurs mathématiques sont ceux habituels hotmis pour le différent (n'est pas égal) s'exprime par l'opérateur « **!=** » (comme en java).
Pour les objets, on utilise « **is** » et « **is not** ».

Exemple de code :

```
if nStr!="" and nRub !="":
```

6.5 - La gestion des erreurs

Il existe une hiérarchie des classes d'exception. Pour le cadre de cette présentation rapide de Python, elle ne sera pas présentée. Comme pour d'autres langages, les mots clefs (**try .. except .. finally**) sont mis en oeuvre :

Exemple de code :

```
    try:
        float(s)
```

```
    return True
  except ValueError:
    return False
```

7 - Interaction avec le système

Le langage Python dispose de nombreuses librairies qui permettent notammant d'interagir avec le système (gestion des fichiers, gestion des répertoires ...). Pour cela, dans un module Python, il convient d'importer les modules « **sys** » et « **os** ».

Exemples de code :

```
import os.path
def GetAdress(nTable, nDir):
    rTable = nTable
    if os.path.exists(rTable)== True :
       return rTable
    else:
        retval = os.path.join(nDir,nTable)
        retval = os.path.abspath(retval)

        if os.path.exists(retval):
           return retval
        else:
           return ""

import sys
    reload(sys)
    sys.setdefaultencoding( "iso-8859-1" )
```

Le module « **ctypes** » permet d'accéder aux librairies du système de type **DLL**.

Exemple de code :

```
import ctypes
PathMITAB = CorrigePath(os.path.dirname(__file__))+"mitab.dll"
if os.path.exists(PathMITAB)== True :
   libMITAB = cdll.LoadLibrary(PathMITAB)
```

8 - La programmation orienté objet

8.1 - Le concept objet

Dans les environnements de développement informatique, il a fallu attendre assez longtemps pour voir émerger le concept de l'objet. Son apparition a permis la création de systèmes beaucoup plus complexes mais aussi très empreints de mimétisme. En effet, dans notre monde réel, nous sommes tous entourés d'objets qui ont très souvent deux critères d'appréciation.

8.2 - Le critère descriptif

Ce premier est universel, il contient toutes les caractéristiques qui décrivent l'objet. Nous prendrons comme exemple un dé, si nous avions à le décrire, nous dirions qu'il possède 6 faces avec un chiffre allant de 1 à 6 sur chacune d'elles, que la somme de deux valeurs étant sur des faces opposées vaut 7, que chaque chiffre entre un et six y est repris une et une seule fois, qu'il est (souvent) de couleur rouge et de petite taille. Il serait possible de le décrire plus précisément, mais en réalité, indiquer qu'il est fait de bois, que les nombres sont représentés par une quantité de point qui leur est égal, qu'il dispose de coin arrondi... n'aurait pas été plus éloquent.

8.3 - Le critère d'interaction

Le deuxième critère est celui d'interaction, il indique l'utilité de l'objet, les possibilités qu'il vous offre. Pour le dé nous pourrions indiquer que celui-ci peut rouler, mais ce n'est pas son rôle. De même, dans certaines circonstances, celui-ci peut vous servir de cale, mais ici encore, nous nous éloignons du sujet. Objectivement, le dé a pour rôle de donner un nombre compris entre son minimum et son maximum (inclus) au hasard. D'ailleurs, on peut ajouter que cela arrive après l'avoir lancé.

8.4 - L'héritage et l'implémentation

Ici, nous avons décrit un objet, et il nous suffit de faire de même en informatique. Mais nous pourrions approfondir la description en indiquant aussi que le dé est en fait dérivé d'un objet de base : le cube. Ainsi nous pourrions dire que le dé :

- est un **cube**.
- est de couleur rouge.
- peut être lancé pour renvoyer un nombre compris entre 1 et 6 (le nombre de face qui le compose).

puis expliquer que le cube :

- est un volume géométrique à trois dimensions.
- est constitué de 6 **carrés**.

puis bien sûr qu'un **carré** :

- est une figure géométrique à deux dimensions.

Et nous pourrions continuer précisant le terme **dimension** mais dans notre cas ce n'est pas utile. Nous pouvons ainsi établir le schémas suivant : le dé hérite des caractéristiques du cube (c'est un cube). Mais on ne peut pas dire que le cube hérite des caractéristiques du carré. En effet, on indique bien qu'il est **constitué de** mais pas qu'il **est** et c'est la toute la différence, vous êtes constitué de deux bras musclés mais vous n'êtes pas deux bras musclés (sauf si vous êtes déménageur... *c'est une blague bien entendu, les déménageurs sont suffisamment allègre pour ne pas lancer un avis de recherche sur ma tête*) ! Nous dirons donc que :

- l'objet **cube** implémente l'objet **surface carré**
- **l'objet dé hérite de l'objet cube**

9 - Démarrer le développement d'un plugin Python pour QGIS

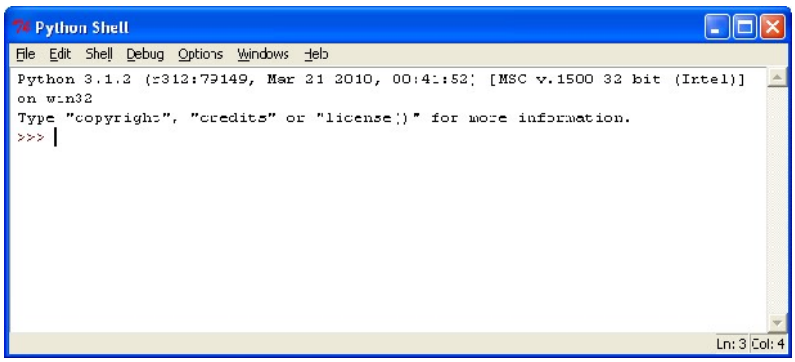
9.1 - L'interpréteur et son fonctionnement

Python est un langage de programmation interprété multi-paradigme. Il favorise la programmation impérative structurée, et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl.

Dans certains langages, le code source n'est pas préalablement traduit en langage machine par un compilateur. Dans ce cas, la transformation en langage machine se fait au moment de l'exécution du programme : un interpréteur traduit le programme, ligne par ligne.

Comme pour les langages compilés, il est nécessaire de disposer d'un interpréteur approprié pour chaque langage utilisé. Un programme écrit en langage Python doit être traité par un interpréteur Python.

Quand une ligne du programme doit être exécutée un grand nombre de fois, l'interpréteur la traduit autant de fois qu'elle est exécutée. Il en résulte une perte de temps et donc une moins grande rapidité des programmes en langage interprété par rapport aux langages compilés.



10 - Développer un plugin (extension) pour QGIS

10.1 - Les éléments structurants

Une extension présente une IHM qui comporte au minimum un menu, et parfois une barre d'outils associée, des formulaires (fichiers « ui » réalisés avec « **PyQt** » ou « **SDK Qt** », cf. paragraphe « ***les principaux outils de développement*** »).

Tous ces éléments sont définis, initialisés dans le module principal de l'extension (nom libre, parfois « main.py » à défaut). La porte d'entrée pour QGIS est toujours un fichier « **__init__.py** ». Ce dernier présente un contenu du type suivant :

Exemple de code :

```
#import de librairie Python ou autres si nécessaire
import ConfigParser
import os.path
p = ConfigParser.ConfigParser()

#dans cet exemple, les données identitaires de l'extension ne sont
#pas stockées en dur dans un module mais lues dans un fichier « ini »

here = os.path.join(os.path.dirname(__file__),"config.ini")
p.read(here)

def name():
   #ci-dessous lecture ou retour valeur
   #les deux modes sont présents ici pour l'exemple !!!!
  return p.get('general','name')
  return "Mon nom"

def authorName():
  return p.get('general','authorName')

def description():
  return p.get('general','description')

def version():
  return p.get('general','version')

def qgisMinimumVersion():
  return p.get("general","qgisMinimumVersion")

#La classFactory, constructeur de l'extension, indique le nom
#du module à lancer. Ici « Main » et la classe à initialiser « MainPlugin »
def classFactory(iface):
  from main import MainPlugin
  return MainPlugin(iface)
```

Le contenu du « **config.ini** », simple fichier texte, est toujours le suivant :

Exemple de code :

```
[general]
name: Nom de l'extension
version: Numéro de version
description: Texte de présention
author: Nom(s) du ou des auteurs
qgisMinimumVersion: version de QGIS à partir de laquelle l'extension est supportée.
```

Toutes ces informations sont présentées dans le gestionnaire d'extension et par l'outil de récupération des extensions.
Le contenu du module « **Main** » dans notre exemple comportera dans sa partie haute les éléments suivants :

Exemple de code :

```
class MainPlugin(object):
  def __init__(self, iface):
    self.name = "MenuName"
    self.iface = iface

#ci-dessous, on essaie de récupérer la version de QGIS
    try:
      QGISVersionID = int(unicode( QGis.QGIS_VERSION_INT ))
    except:
      QGISVersionID = int(unicode( QGis.qgisVersion )[ 0 ])

  def startRender(self, context):
    #un traitement sinon ...
    pass

  def stopRender(self, context):
    pass

  def initGui(self):
    #on démarre la construction de l'environnement plugin
    self.menu=QMenu("Titre Menu")
    self.MenuItem1 = Qaction("Que faire ? ...", self.iface.mainWindow())
    QObject.connect(self.MenuItem1 ,SIGNAL("triggered()"),self.MyAction1)
    #dans la ligne ci-dessus, on créé une ligne de commande MenuItem1 au menu
    #Le « trigger » est le déclencheur, le signal. Ce signal est capté sur l'élément du menu  MenuItem1
    #attention à la casse !

    #un élement case à cocher
    self. MenuItem4 = QAction( "Texte de la case",self.iface.mainWindow( ) )
    self.MenuItem4.setCheckable(True)
    self.MenuItem4.setChecked(False)
    QObject.connect(self.MenuItem4 ,SIGNAL("triggered()"),self.MyAction4)


    #construction du menu à proprement parler
```

```
self.menu.addAction(self.MenuItem1)
self.menu.addSeparator()
self.menu.addActions([self.MenuItem2, self.MenuItem3])

#pour une barre d'outils
toolbar = self.iface.addToolBar(self.toolBarName)
self._toolbar = toolbar
toolbar.addAction(self.MenuItem1)
toolbar.addAction(self.MenuItem2)
...
```

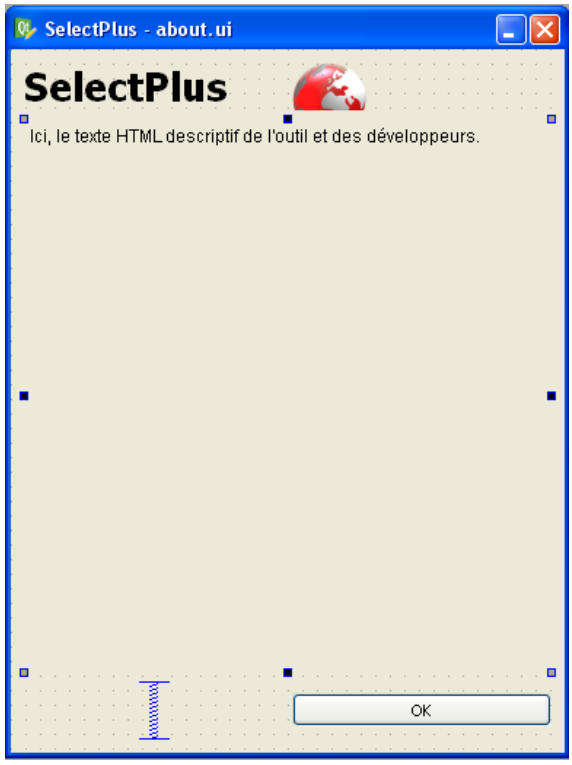
Notez que les méthodes ou fonctions de la classe « **MainPlugin** » sont indentées avec un retrait. Les fonctions ou procédures sans retrait ne sont pas uniquement utilisables par l'objet.

10.2 - La création d'interfaces pour QGIS et leur implémentation

On peut démarrer avec la création d'une boîte de dialogue « A propos ... ». La création de la « base » peut se faire sous « **PyQt (Qt Designer)** » ou le « **SDK Qt** » (ou d'autres). Cette création aboutie à générer un fichier « **ui** ».

Exemple :

Le fichier « **ui** » est le suivant :



Cette « base » est alimentée par des scripts Python placés dans des modules « **doAbout.py** » et « **About.py** ».

Le fichier « **doAbout.py** » contient le constructeur de classe objet, et le fichier « **About.py** » le contenu de la boîte de dialogue pour sa restitution telle que présentée ci-dessus.

Exemple de code :

#contenu de « **doAbout.py** »

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from qgis.core import *
from about import Ui_Dialog

class Dialog(QDialog, Ui_Dialog):
        def __init__(self):
                QDialog.__init__(self)
                self.setupUi(self)
```

#contenu de « **About.py** »

```
# -*- coding: utf-8 -*-
# -*- coding: iso-8859-1 -*-
import os.path
from PyQt4 import QtCore, QtGui

class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(QtCore.QSize(QtCore.QRect(0,0,330,350).size()).expandedTo(Dialog.minimumSizeHint()))
```

```
self.gridlayout = QtGui.QGridLayout(Dialog)
self.gridlayout.setObjectName("gridlayout")

self.label_2 = QtGui.QLabel(Dialog)
self.labelImage = QtGui.QLabel(Dialog)
myPath = os.path.dirname(__file__)+"/selectplus.png";
myDefPath = myPath.replace("\\","/");
carIcon = QtGui.QImage(myDefPath) #"python/plugins/selectplus/selectplus.png")
self.labelImage.setPixmap(QtGui.QPixmap.fromImage(carIcon))

font = QtGui.QFont()
font.setPointSize(15) #20
font.setWeight(50) #75
font.setBold(True)
self.label_2.setFont(font)
self.label_2.setTextFormat(QtCore.Qt.RichText)
self.label_2.setObjectName("label_2")
self.gridlayout.addWidget(self.label_2,1,1,1,2)
self.gridlayout.addWidget(self.labelImage,1,5,1,2)

self.textEdit = QtGui.QTextEdit(Dialog)

palette = QtGui.QPalette()

brush = QtGui.QBrush(QtGui.QColor(0,0,0,0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active,QtGui.QPalette.Base,brush)

brush = QtGui.QBrush(QtGui.QColor(0,0,0,0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive,QtGui.QPalette.Base,brush)

brush = QtGui.QBrush(QtGui.QColor(255,255,255))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled,QtGui.QPalette.Base,brush)
self.textEdit.setPalette(palette)
self.textEdit.setAutoFillBackground(True)
self.textEdit.width = 320
self.textEdit.height = 380
self.textEdit.setFrameShape(QtGui.QFrame.NoFrame)
self.textEdit.setFrameShadow(QtGui.QFrame.Plain)
self.textEdit.setReadOnly(True)
self.textEdit.setObjectName("textEdit")

self.textEdit.setTextInteractionFlags(QtCore.Qt.TextBrowserInteraction)
#QtCore.Qt.LinksAccessibleByMouse|QtCore.Qt.TextSelectableByKeyboard

self.gridlayout.addWidget(self.textEdit,2,1,5,2) #2,1,1,2

self.pushButton = QtGui.QPushButton(Dialog)
self.pushButton.setObjectName("pushButton")
self.gridlayout.addWidget(self.pushButton,4,2,1,1)

spacerItem = QtGui.QSpacerItem(20,40,QtGui.QSizePolicy.Minimum,QtGui.QSizePolicy.Expanding)
self.gridlayout.addItem(spacerItem,3,1,1,1)
```

```
        self.retranslateUi(Dialog)
        QtCore.QObject.connect(self.pushButton,QtCore.SIGNAL("clicked()"),Dialog.reject)
        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def retranslateUi(self, Dialog):
        Dialog.setWindowTitle(QtGui.QApplication.translate("Dialog", "Outils de sélection", None,
QtGui.QApplication.UnicodeUTF8))
        self.label_2.setText(QtGui.QApplication.translate("Dialog", "Outils de sélection 0.8", None,
QtGui.QApplication.UnicodeUTF8))
        self.textEdit.setHtml(QtGui.QApplication.translate("Dialog", "<html><head><meta name=\"qrichtext\"
content=\"1\" /><style type=\"text/css\">\n"
        "p, li { white-space: pre-wrap; }\n"
        "</style></head><body style=\" font-family:\'Sans Serif\'; font-size:8pt; font-weight:400; font-
style:normal;\">\n"
        "<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-
indent:0px; font-family:\'MS Shell Dlg 2\'; font-size:8pt;\"><span style=\" font-weight:600;\">Outils de
sélection</span> est une collection d'outils de sélection sur les objets vecteurs développée par <b>Barry
Rowlingson</b>. Ils ne sont pas une partie du moteur de Qgis et tout probleme ne peut être adressé aux
développeurs QGIS.</p>\n"
        "<p style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px;
-qt-block-indent:0; text-indent:0px;\"></p>\n"
        "<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-
indent:0px;\"><a href=\"barry.rowlingson@gmail.com\">Barry ROWLINGSON</a><br>Développement originel
(2008)."
        "<br><a href=\"#\">Carson FARMER</a><br>Prise en compte des projections (31 mars 2009)."
        "<br><a href=\"christophe.masse@developpement-durable.gouv.fr\">Christophe MASSE</a><br>DO-Sud
Est - CP2I - <b>MEEDDM</b>.<br>Multi-couches, cible, toolbar et Iconographie (1<sup>er</sup> juillet
2010).</p></body></html>", None, QtGui.QApplication.UnicodeUTF8))
        self.pushButton.setText(QtGui.QApplication.translate("Dialog", "OK", None,
QtGui.QApplication.UnicodeUTF8))
```

Notez aussi qu'il est possible d'implémenter ou d'enrichir directement les interfaces de 'IHM QGIS. Ci-dessous un exemple simple de classe « Widget » pour le paramétrage d'une classe « MapInfoFillSymbolLayer » :

Exemple de code :

```
class MapInfoFillSymbolLayerWidget(QgsSymbolLayerV2Widget):
  def __init__(self, parent=None):
    QgsSymbolLayerV2Widget.__init__(self, parent)
    self.layer = None
    self.label = QLabel("Couleur:")
    self.labelInfos = QLabel("Ceci n'est qu'un exemple d'interface pour la sélection de la couleur ...")
    self.spinColor = QColorDialog() #QLabel(str(QColor(self.layer.color)))
    self.hbox = QHBoxLayout()
    self.hbox.addWidget(self.label)
    self.hbox.addWidget(self.labelInfos)
    self.hbox.addWidget(self.spinColor)
    self.setLayout(self.hbox)
    self.connect( self.spinColor, SIGNAL("valueChanged(long)"), self.colorChanged)

  def setSymbolLayer(self, layer):
```

```
  if layer.layerType() != "MapInfoBrush":
    return
  self.layer = layer
  self.spinColor.setValue(layer.color)

def symbolLayer(self):
  return self.layer

def colorChanged(self, value):
  self.layer.color = value
  self.emit(SIGNAL("changed()"))

def colorSelected(self, value):
  self.spinColor = QColorDialog.getColor(value, self, "ColorDialog")
```

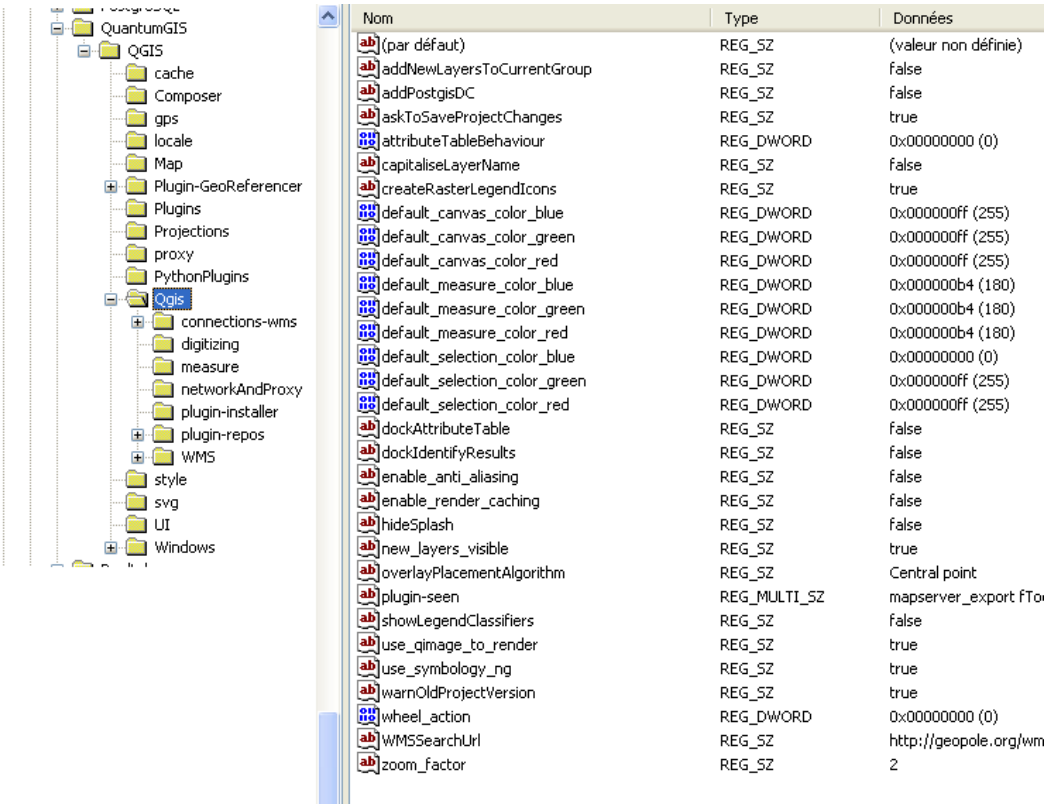
10.3 - Obtenir des informations sur le paramétrage de QGIS

Il peut être intéressant de récupérer les informations sur le paramétrage du logiciel et sur les informations du profil utilisateur.

Exemple de code :

```
nSymbologyV2 = QSettings().value( "Qgis/use_symbology_ng", QVariant( "" ) ).toBool()
```

La classe « **QSettings** » de l'API de QGIS permet d'obtenir les informations sur le paramétrage du logiciel qui sont stockées dans la base de registre (sous Windows) :



Le plus souvent, la récupération de la valeur de la clef s'accompagne d'une opération de conversion (toBool(), toString(), toUInt()[0], toStringList() ...)

Exemple de code :

```
seenPlugins = QSettings().value(seenPluginGroup, Qvariant(QStringList(self.mplugins.keys()))).toStringList()
```

De même, il est possible de changer ou de passer des valeurs de paramétrage.

Exemple de code :

```
QSettings().setValue(key+"/valid", QVariant(True))
```

Ci-dessous, un extrait de « **installer_data.py** » :

```
# --- class QPHttp  ------------------------------------------------------------------- #
# --- It's a temporary workaround for broken proxy handling in Qt ----------------------- #
class QPHttp(QHttp):
  def __init__(self,*args):
    QHttp.__init__(self,*args)
    settings = QSettings()
    settings.beginGroup("proxy")
    if settings.value("/proxyEnabled").toBool():
      self.proxy=QNetworkProxy()
      proxyType = settings.value( "/proxyType", QVariant(0)).toString()
      if len(args)>0 and settings.value("/proxyExcludedUrls").toString().contains(args[0]):
        proxyType = "NoProxy"
      if proxyType in ["1","Socks5Proxy"]: self.proxy.setType(QNetworkProxy.Socks5Proxy)
      elif proxyType in ["2","NoProxy"]: self.proxy.setType(QNetworkProxy.NoProxy)
      elif proxyType in ["3","HttpProxy"]: self.proxy.setType(QNetworkProxy.HttpProxy)
      elif proxyType in ["4","HttpCachingProxy"] and QT_VERSION >= 0X040400:
self.proxy.setType(QNetworkProxy.HttpCachingProxy)
      elif proxyType in ["5","FtpCachingProxy"] and QT_VERSION >= 0X040400:
self.proxy.setType(QNetworkProxy.FtpCachingProxy)
      else: self.proxy.setType(QNetworkProxy.DefaultProxy)
      self.proxy.setHostName(settings.value("/proxyHost").toString())
      self.proxy.setPort(settings.value("/proxyPort").toUInt()[0])
      self.proxy.setUser(settings.value("/proxyUser").toString())
      self.proxy.setPassword(settings.value("/proxyPassword").toString())
      self.setProxy(self.proxy)
    settings.endGroup()
    return None
# --- /class QPHttp  ------------------------------------------------------------------- #
```

10.4 - Utilisation de l'API QGIS

L'API QGIS est très riche. Si la documentation technique est assez bien documentée, pour trouver des exemples concrets de mise en application, le mieux est le plus souvent de sonder les outils à disposition. En effet, toutes les extensions sous « **Python** » sont parfaitement lisibles avec un simple éditeur de texte. Mieux vaut néanmoins installer l'« **IDLE** » (« Integrated Development Environment ») ou utiliser un éditeur style « **NotePad ++** » qui jouera la colorisation syntaxique du langage Python. Notez que QGIS embarque un environment d'exécution Python 2.5 . Les développements sur « **SELECTPLUSFR** » et « **OPENWOR** » ont été réalisés sous Python 3.1, sans problèmes de compatibilité rencontrée à leur exécution sous QGIS.

Ci-dessous, la classification hiérarchique de l'API issue du site http://doc.qgis.org/head/hierarchy.html :

GEOSException
- GEOSInit
- HalfEdge
- Line3D
- MyLine
- Node
- ParametricLine
  - Bezier3D
- Point3D
- QGis
- QgisInterface
- QgisPlugin
  - QgsRendererPlugin
  - QgsVectorOverlayPlugin
- QgsAction
- QgsApplication
- QgsApplyDialog
- QgsAttributeAction
- QgsAttributeEditor
- QgsClipper
- QgsColorBrewerPalette
- QgsColorButton
- QgsColorButtonV2
- QgsColorRampShader::ColorRampItem
- QgsComposerItem
  - QgsComposerArrow
  - QgsComposerItemGroup
  - QgsComposerLabel
  - QgsComposerLegend
  - QgsComposerMap
  - QgsComposerPicture
  - QgsComposerScaleBar
  - QgsComposerShape
  - QgsComposerTable

        - QgsComposerAttributeTable
        - QgsComposerTextTable
    - QgsPaperItem
- QgsComposerLegendItem
    - QgsComposerGroupItem
    - QgsComposerLayerItem
    - QgsComposerRasterSymbolItem
    - QgsComposerSymbolItem
    - QgsComposerSymbolV2Item
- QgsComposerView
- QgsComposition
- QgsContextHelp
- QgsContrastEnhancement
- QgsContrastEnhancementFunction
    - QgsClipToMinMaxEnhancement
    - QgsLinearMinMaxEnhancement
    - QgsLinearMinMaxEnhancementWithClip
- QgsCoordinateReferenceSystem
- QgsCoordinateTransform
- QgsCredentials
    - QgsCredentialDialog
    - QgsCredentialsConsole
- QgsDataProvider
    - QgsRasterDataProvider
    - QgsVectorDataProvider
- QgsDataSourceURI
- QgsDetailedItemData
- QgsDetailedItemDelegate
- QgsDetailedItemWidget
- QgsDistanceArea
- QgsEncodingFileDialog
- QgsException
    - QgsCsException
- QgsFeature
- QgsFeatureRendererV2
    - QgsCategorizedSymbolRendererV2
    - QgsGraduatedSymbolRendererV2
    - QgsRuleBasedRendererV2
    - QgsSingleSymbolRendererV2
- QgsField
- QgsFileDropEdit
- QgsGenericProjectionSelector
- QgsGeometry
- QgsGeometry::Error
- QgsGeometryAnalyzer
- QgsGridFileWriter
- QgsHttpTransaction
- QgsInterpolator
    - QgsIDWInterpolator

- QgsTINInterpolator
- QgsInterpolator::LayerData
- QgsLabel
- QgsLabel::labelpoint
- QgsLabelAttributes
- QgsLabelCandidate
- QgsLabelingEngineInterface
  - QgsPalLabeling
- QgsLegendInterface
- QgsLegendModel
- QgsLogger
- QgsLongLongValidator
- QgsLUDialog
- QgsMapCanvas
- QgsMapCanvas::CanvasProperties
- QgsMapCanvasItem
  - QgsAnnotationItem
    - QgsFormAnnotationItem
    - QgsTextAnnotationItem
  - QgsRubberBand
  - QgsVertexMarker
- QgsMapCanvasLayer
- QgsMapCanvasMap
- QgsMapCanvasSnapper
- QgsMapLayer
  - QgsPluginLayer
  - QgsRasterLayer
  - QgsVectorLayer
- QgsMapLayerRegistry
- QgsMapOverviewCanvas
- QgsMapRenderer
- QgsMapTip
- QgsMapTool
  - QgsMapToolEmitPoint
  - QgsMapToolPan
  - QgsMapToolZoom
- QgsMapToPixel
- QgsMarkerCatalogue
- QgsMessageOutput
  - QgsMessageOutputConsole
  - QgsMessageViewer
- QgsNetworkAccessManager
- QgsNineCellFilter
  - QgsDerivativeFilter
    - QgsAspectFilter
    - QgsSlopeFilter
  - QgsRuggednessFilter
  - QgsTotalCurvatureFilter
- QgsOverlayAnalyzer

- QgsOverlayObject
- QgsOverlayObjectPositionManager
  - QgsCentralPointPositionManager
  - QgsPALObjectPositionManager
- QgsPALGeometry
- QgsPalGeometry
- QgsPalLayerSettings
- QgsPanningWidget
- QgsPluginLayerRegistry
- QgsPluginLayerType
- QgsPoint
- QgsProject
- QgsProject::Imp
- QgsProjectBadLayerHandler
  - QgsProjectBadLayerDefaultHandler
  - QgsProjectBadLayerGuiHandler
- QgsProjectFileTransform
- QgsProjectFileTransform::transform
- QgsProjectionSelector
- QgsProjectVersion
- QgsProperty
  - QgsPropertyKey
  - QgsPropertyValue
- QgsProviderCountCalcEvent
- QgsProviderExtentCalcEvent
- QgsProviderMetadata
- QgsProviderRegistry
- QgsQuickPrint
- QgsRasterBandStats
- QgsRasterImageBuffer
- QgsRasterPyramid
- QgsRasterShader
- QgsRasterShaderFunction
  - QgsColorRampShader
  - QgsFreakOutShader
  - QgsPseudoColorShader
- QgsRasterTransparency
- QgsRasterTransparency::TransparentSingleValuePixel
- QgsRasterTransparency::TransparentThreeValuePixel
- QgsRasterViewPort
- QgsRectangle
- QgsRenderContext
- QgsRenderer
  - QgsContinuousColorRenderer
  - QgsGraduatedSymbolRenderer
  - QgsSingleSymbolRenderer
  - QgsUniqueValueRenderer
- QgsRendererCategoryV2
- QgsRendererRangeV2

- QgsRendererV2AbstractMetadata
  - QgsRendererV2Metadata
- QgsRendererV2Registry
- QgsRuleBasedRendererV2::Rule
- QgsRunProcess
- QgsScaleBarStyle
  - QgsDoubleBoxScaleBarStyle
  - QgsNumericScaleBarStyle
  - QgsSingleBoxScaleBarStyle
  - QgsTicksScaleBarStyle
- QgsScaleCalculator
- QgsSearchString
- QgsSearchTreeNode
- QgsSearchTreeValue
- QgsSnapper
- QgsSnapper::SnapLayer
- QgsSnappingResult
- QgsStyleV2
- QgsSymbol
- QgsSymbolLayerV2
  - QgsFillSymbolLayerV2
    - QgsSimpleFillSymbolLayerV2
    - QgsSVGFillSymbolLayer
  - QgsLineSymbolLayerV2
    - QgsLineDecorationSymbolLayerV2
    - QgsMarkerLineSymbolLayerV2
    - QgsSimpleLineSymbolLayerV2
  - QgsMarkerSymbolLayerV2
    - QgsFontMarkerSymbolLayerV2
    - QgsSimpleMarkerSymbolLayerV2
    - QgsSvgMarkerSymbolLayerV2
- QgsSymbolLayerV2AbstractMetadata
  - QgsSymbolLayerV2Metadata
- QgsSymbolLayerV2Registry
- QgsSymbolLayerV2Utils
- QgsSymbologyV2Conversion
- QgsSymbolV2
  - QgsFillSymbolV2
  - QgsLineSymbolV2
  - QgsMarkerSymbolV2
- QgsSymbolV2LevelItem
- QgsSymbolV2RenderContext
- QgsTolerance
- QgsUndoCommand
- QgsUndoCommand::AttributeChangeEntry
- QgsUndoCommand::GeometryChangeEntry
- QgsVector
- QgsVectorColorRampV2
  - QgsVectorColorBrewerColorRampV2

    - QgsVectorGradientColorRampV2
    - QgsVectorRandomColorRampV2
- QgsVectorDataProvider::NativeType
- QgsVectorFileWriter
- QgsVectorLayer::RangeData
- QgsVectorOverlay
- QgsZonalStatistics
- TriangleInterpolator
    - CloughTocherInterpolator
    - LinTriangleInterpolator
- Triangulation
    - DualEdgeTriangulation
    - TriDecorator
        - NormVecDecorator
- Vector3D
- vertexData

Ci-dessous un exemple d'utilisation de l'API :

Exemple de code :

```
class InvertAllAction(SelectAction):
    def __init__(self,iface):
        SelectAction.__init__(self,iface,"Inverser toutes les sélections","Inverser toutes les sélection",
"invselall.png",0)
        return None
    def doit(self):
        nLayers = self.iface.mapCanvas().layerCount()
        QApplication.setOverrideCursor( QCursor( Qt.WaitCursor ) )
        for i in range(0, nLayers):
            layer = self.iface.mapCanvas().layer(i)
            if layer.isValid():
              if layer.type() == layer.VectorLayer:
                all = layerIds(layer)
                current = layer.selectedFeaturesIds()
                new = list(set(all)-set(current))
                layer.setSelectedFeatures(new)
        else:
            ActualiseMSg(self)
            return None

        QApplication.restoreOverrideCursor()
        return None
```

La liste ne reflète pas intégralement les possibilités de l'API QGIS : certains élements ne sont pas présentés.

Exemple de code :

```
zQtLine = QgsSymbologyUtils.qString2PenStyle(tLine[zLine])
zQtBrush = QgsSymbologyUtils.qString2BrushStyle(tBrush[zBrush])
```

Parfois, il faut aller au plus près du coeur source pour trouver certaines informations (valeurs paramètres de fonctions)

Exemple de code :

```
 def clickInfosEtiqPOS(self):
     zCodes = "Aligné horizontalement au centre :" + str(QtCore.Qt.AlignHCenter) + "\n"
     zCodes = zCodes + "Aligné horizontalement à droit :" + str(QtCore.Qt.AlignRight) + "\n"
     zCodes = zCodes + "Aligné horizontalement à gauche :" + str(QtCore.Qt.AlignLeft) + "\n\n"
     zCodes = zCodes + "Aligné verticalement au centre :" + str(QtCore.Qt.AlignVCenter) + "\n"
     zCodes = zCodes + "Aligné verticalement en bas :" + str(QtCore.Qt.AlignBottom) + "\n"
     zCodes = zCodes + "Aligné verticalement en haut :" + str(QtCore.Qt.AlignTop) + "\n"
     QMessageBox.information(None, "Codes alignement : ", zCodes)


 tLine = {'1':'NoPen', '2':'SolidLine', '3':'DashLine', '4':'DotLine',
                          '5':'DashDotLine', '6':'DashDotDotLine'
                          }

tBrush = {'1':'NoBrush', '2':'SolidPattern', '3':'HorPattern', '4':'VerPattern',
                          '5':'BDiagPattern', '6':'FDiagPattern','7':'CrossPattern','8':'DiagCrossPattern',
                          '12':'Dense1Pattern', '13':'Dense2Pattern','14':'Dense3Pattern','15':'Dense4Pattern',
                          '16':'Dense5Pattern', '17':'Dense6Pattern','18':'Dense7Pattern','19':'HorPattern',
                          '20':'HorPattern', '24':'VerPattern','25':'VerPattern','29':'BDiagPattern',
                          '30':'BDiagPattern', '34':'FDiagPattern','35':'FDiagPattern','39':'CrossPattern','40':'CrossPattern'
                          }
```

Enfin, le développement « entre » deux versions peut singulièrement compliqué la tâche.

Mais, au regard de l'API de QGIS, de la richesse du langage Python, une fois ce dernier maîtrisé, des librairies et outils dédiés (« Qt ») (cf. paragraphe « Les principaux outils de développement »),, le développement d'extensions offre des possibilités extrêmement importantes et performantes. D'autant que contrairement à d'autres logiciels (« MapInfo » et « MapBasic »), les IHM des extensions bénéficient de la simplicité de mise en oeuvre des formulaires « **ui** ».

10.5 - L'internationalisation sous QGIS

Il est possible d' « internationaliser » son extension (au moins prévoir les appels dans le code). Mais la traduction semble fonctionner sous le même principe que les fichiers « **mo** » et « **po** » sous « PHP ». A savoir que dans l'absolu, il faut que la traduction de votre extension soit implémentée sous QGIS (association des mots clefs avec la valeur en retour de la traduction).

Exemple de code :

```
QtCore.QCoreApplication.translate("MainPlugin", "Hello")

        Dialog.setWindowTitle(QtGui.QApplication.translate("Dialog", "Outils de sélection", None,
QtGui.QApplication.UnicodeUTF8))
        self.label_2.setText(QtGui.QApplication.translate("Dialog", "Outils de sélection 0.8", None,
QtGui.QApplication.UnicodeUTF8))
```

11 - Quelques exemples de code

11.1 - Manipulations sur les formats « QStringList » de Qt et « QgsStringMap » de l'API QGIS

Certaines fonctions de l'API QGIS retournent sont les informations dans ces formats.

Le format « QStringList » peut être utilisé comme une liste (hérite de « QList<QString>) ». Il est possible d'obtenir sa dimension (fonction « LEN ») et de le parcourir (exemple ci-dessous) ou d'atteindre directement un élément par un pointeur.

Exemple de code :

```
MyList = QgsRendererV2Registry().renderersList()
toto = "\n"
for i in range(len(MyList)):
    toto = toto + str(MyList[i])+"\n"
```

Pour créer une variable de type « QStringList », on initialise la variable (« QStringList() »), puis pour ajouter des éléments, on utilise la synthaxe « **<<** ».

Exemple de code :

```
zQStringList = QStringList()
 for i in range(len(ssLayer)):
        zQStringList <<   ssLayer[i]
```

Pour une liste classique, les appels sont un peu différents : la synthaxe « **{}** » permert d'initialiser la liste. L'ajout d'éléments peut se faire par la méthode « **append** » ou par pointeur. Les méthodes « remove » permettent de jouer les suppressions. La classe « QList » comporte aussi une méthode pour compter les occurrences sur une valeur.

Exemple de code :

```
symbs = {}
symbs[i] = zValue
 symbs.append(zValue)
```

Le format « **QgsStringMap** » est propre à QGIS. Un exemple est donnée ci-dessous de ce type de structuration de l'information. Les accolades marquent le début et la fin de la variable. Chaque membre est séparé par une virgule. Chaque membre comporte un mot clef et une valeur associée (un peu comme les dictionnaires) séparés par le caractère « : ».

Exemple de code :

```
{QString("svgFilePath:") : Qstring(PathTexture), QString("width:") : QString(float(20)) }
```

Ci-dessous, un exemple de manipulations sur ce type de variable :

Exemple de code :

```
MyMap = QgsStringMap()
MyMap.insert("color", QString(QColor(0,255,0)))
MyMap.insert("style", QString("verpattern"))
Map.insert("color_border", QString(QColor(0,0,0)))
MyMap.insert("style_border", QString("dashline"))
MyMap.insert("width_border", QString(0.26))
```

11.2 - Créer une Annotation

Ci-dessous, un exemple de manipulations sur ce besoin :

Exemple de code :

```
textItem = QgsTextAnnotationItem( self.iface.mapCanvas())
X = float(572703)
Y = float(6575568)
point = QgsPoint(X, Y)
textItem.setMapPosition(point)
textItem.setFrameSize(QSizeF(200,50))
textItem.setFrameColor(QColor(0,0,155))
textItem.setFrameBackgroundColor(QColor(200,200,200))
zDoc = QtextDocument()
zText = "<b>Essai</b>"
zDoc.setHtml(str("<font style=\"color:"+str(zForeColor)+"; font-family:'"+str(zFont)+"'; font-size: " +str(zFontSize)
+"px\">"+str(zText)+"</font>"))
textItem.setDocument(zDoc)
```

Notez qu'il est tout à fait possible de personnaliser le rendu du texte bien au-delà de ce que l'IHL « Annotations » propose (gras, italique), puisqu'un certain nombre de balises HTML est supporté.

11.3 - Parcourir les données attributaires d'une couche

Il peut être intéressant de parcourir l'intégralité des données attributaires d'une couche, mais le plus souvent on cherche à atteindre les données d'un enregistrement. L'exemple ci-dessous présente ce cas de figure avec l'usage de la méthode « **featureAtId** » de la classe « **provider** ». « **feat** » n'est que le conteneur de stockage des données retournées.

Exemple de code :

```
vprovider = zLayer.dataProvider()
feat = QgsFeature()
index = vprovider.fieldNameIndex(nLabelFieldName)
vprovider.featureAtId(zID,feat,True,[index])
geom = feat.geometry()
X = float(geom.centroid().asPoint().x())
Y = float(geom.centroid().asPoint().y())
```

Une boucle sur l'intégralité d'un jeu de données sera de la forme ci-dessous. Ici, pour déterminer les valeurs minimum et maximum des données (la fonction « **GetMinMax** » retourne le couple).

Exemple de code :

```
def GetMinMax(zLayer, zIndexField):
    vprovider = zLayer.dataProvider()
    allAttrs = vprovider.attributeIndexes()
    vprovider.select( allAttrs )
    fields = vprovider.fields()
    minVal = 0
    maxVal = 0

    feat = QgsFeature()
    first = True

    nFeat = vprovider.featureCount()
    while vprovider.nextFeature( feat ):
        atMap = feat.attributeMap()
        value = float( atMap[ zIndexField ].toDouble() [ 0 ] )
        if first:
                minVal = value
          maxVal = value
                first = False
            else:
                if value < minVal: minVal = value
                if value > maxVal: maxVal = value


    return (minVal, maxVal)
```

11.4 - Créer des points

L'exemple ci-dessous montre comment créer des points issus des centroïdes des objets géographiques parcourus.

Exemple de code :

```
zID = 0
provider = layer.dataProvider()
allA=provider.attributeIndexes()
provider.select(allA)
feat = QgsFeature()

Centro = QgsVectorLayer("Point", "Centroide_"+QString.fromLocal8Bit(layer.name()), "memory")
prCentro = Centro.dataProvider()
    ret = prCentro.addAttributes( [ QgsField("Etiquette", QVariant.String) , QgsField("Valeur", QVariant.Int),
QgsField("CoordX", QVariant.Double), QgsField("CoordY", QVariant.Double)] )
fet = QgsFeature()
zID = 0

while provider.nextFeature(feat):
    geom = feat.geometry()
    attrs = feat.attributeMap()
    fet.setGeometry(QgsGeometry(geom.centroid()))
    fet.addAttribute(0, QVariant("Centroide " + str(zID)))
    fet.addAttribute(1, QVariant(zID))
    fet.addAttribute(2, QVariant(geom.centroid().asPoint().x()))
    fet.addAttribute(3, QVariant(geom.centroid().asPoint().y()))
    prCentro.addFeatures( [ fet ] )
    zID = zID + 1

zCentroideLayer = QgsMapLayerRegistry.instance().addMapLayer(Centro)
```

En fin de traitement, la couche temporaire « **Centro** » est ajoutée à la fenêtre carte.

11.5 - Modifier l'aspect du curseur de la souris pour indiquer un traitement en cours

Exemple de code :

```
#Au lancement du traitement
QApplication.setOverrideCursor( QCursor( Qt.WaitCursor ) )
#Fin du traitement
QApplication.restoreOverrideCursor()
```

11.6 - Obtenir les ID des objets d'une couche

Le code ci-dessous montre comment récupérer les ID des objets d'une couche et les placer dans un tableau. « **ids** ». On retrouve la méthode « **append** » pour l'ajout d'éléments au tableau.

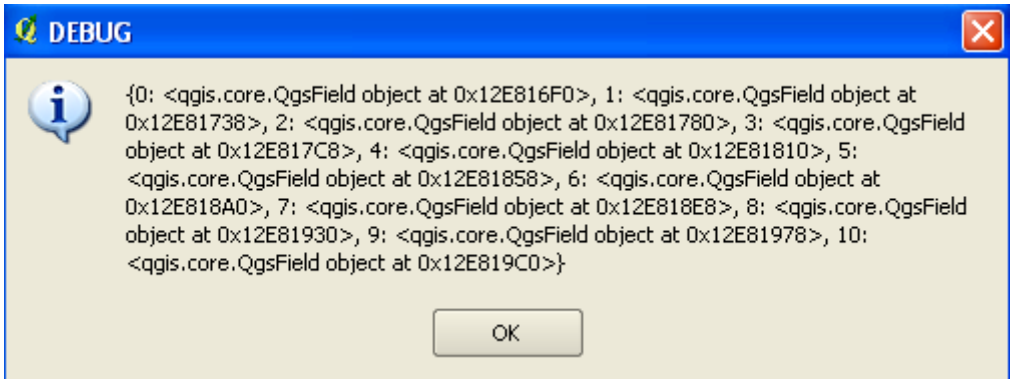
Exemple de code :

```
def layerIds(layer):
    ids = []
    p = layer.dataProvider()
    allAttrs = p.attributeIndexes()
    p.select(allAttrs)
    f = QgsFeature()
    while p.nextFeature(f):
        ids.append(f.id())
    return ids
```

11.7 - Obtenir la liste des champs d'une couche

Exemple de code :

```
def getFieldList( vlayer ):
    vprovider = vlayer.dataProvider()
    feat = QgsFeature()
    allAttrs = vprovider.attributeIndexes()
    vprovider.select( allAttrs )
    myFields = vprovider.fields()
    return myFields
```



Dans cette forme basique, cette fonction retourne une information structurée de type

« **QgsStringMap** ». De plus, les « valeurs » stockées sont en fait des références aux objets champs. Pour une meilleure lisibilité des informations, un post-traitement est indispensable.

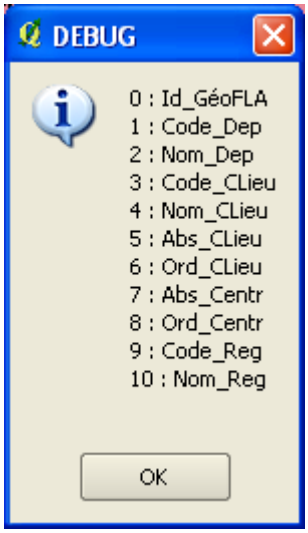
11.8 - Lire le contenu d'un tableau

A partir de l'exemple, précédent, le parcour d'un tableau ne se fait pas à partir des fonctions « **len** » et par pointeur « **index** », mais par les clefs du tableau. Pour rappel, dans notre exemple, les valeurs sont des pointeurs objets « **fields** » (champs). Pour récupérer le nom du champ, il suffit d'invoquer la propriété « name » sur l'objet. Dans l'exemple ci-dessous, nous avons placé une variable de stockage intermédiaire (**nField**), mais l'on peut simplement implémenter : **tabfields[key].name()**.

Exemple de code :

```
tabfields = getFieldList(zLayer)
toto = ""
for key in tabfields.keys():
    nField = (tabfields[key])
    toto = toto + str(key) +" : " + str(nField.name()) +"\n"
QMessageBox.information(None,"DEBUG", str(toto))
```

Ce code retourne le résultat ci-dessous :



Notez que dans l'API de QGIS, certaines fonctions travaillent avec l'index du champ comme paramètre, d'autre avec le nom.

11.9 - Pointer sur une couche par son nom

Exemple de code :

```
def getVectorLayerByName( myName ):
    layermap = QgsMapLayerRegistry.instance().mapLayers()
    for name, layer in layermap.iteritems():
        if layer.type() == QgsMapLayer.VectorLayer and layer.name() == myName:
            if layer.isValid():
                return layer
            else:
                return None
```

Cette fonction place dans la variable « **layer** », la couche qui correspond au nom « **myName** ».

11.10 - Utiliser le « Rubberband » de QGIS (espace dessin temporaire)

Le « **rubberband** » peut être vu comme une zone de représentation temporaire. Elle permet de restituer notamment les actions de sélection par glisser / dessiner (cercles, rectangles, formes libres à main levée ...). La fonction « **reset** » sur un objet « rubberband » permet de nettoyer son contenu.

Exemple de code :

```
self.layer = self.iface.mapCanvas().currentLayer()
rb=QgsRubberBand(self.iface.mapCanvas(),True)
rb.reset()
provider = layer.dataProvider()
allA=provider.attributeIndexes()
provider.select(allA)
feat = QgsFeature()

while provider.nextFeature(feat):
        geom = feat.geometry()
        attrs = feat.attributeMap()
        refX = float(geom.centroid().asPoint().x())
        refY = float(geom.centroid().asPoint().y())
        MyCenter = QgsPoint( refX, refY)
        rb.addPoint(MyCenter)
```

11.11 - Tester si une variable est numérique

Exemple de code :

```
def is_number(s):
    #Test si la chaîne peut être convertie en nombre.
    try:
```

```
        float(s)
        return True
    except ValueError:
        return False
```

11.12 - Lire le contenu d'un fichier texte

L'exemple ci-dessous charge l'intégralité du fichier texte dans une variable.

Exemple de code :

```
myFile = open(nFileURI, 'r')
myInfos = ""
myInfos = myFile.readlines()
myFile.closed
```

On peut aussi dans certains cas opter pour une lecture séquentielle :

Exemple de code :

```
def GetTypeTable(nTable):
    mytable = open(nTable, 'r')
    nType=""
    for line in mytable:
        astring=str(QString.fromLocal8Bit(line))
        astring = astring.upper()
        if astring.find('TYPE')!=-1:
            lst = astring.split()
            nType = NetStrInfos(lst[1], False, False, False, False, ("\""))
            break
    mytable.closed
    if nType == "":
       nType = "NATIVE"
    return nType
```

Mais dans les faits, ce type de lecture peut aussi jouer sur la variable chargée du premier exemple (on la parcourt comme un tableau).

11.13 - Modifier les caractéristiques de la fenêtre légende de QGIS

La fonction « **MakeGroupExpanded** » si-dessous permet d'ouvrir ou de fermer l'intégralité des branches de la légende. La condition « **nCond** » de type booléen indique le sens de l'action.

Exemple de code :

```
def MakeGroupExpanded(self, nCond):
    self.canvas = self.iface.mapCanvas()
    nLayers = self.canvas.layerCount()

    for i in range(0, nLayers):
        self.iface.legendInterface().setGroupExpanded(i, nCond)
```

Les exemples ci-dessous montrent comment changer certaines caractéristiques de la légende :

Exemple de code :

```
#changer le nom affiché
zLayer.setLayerName(zNewLayerName)
#rafraîchir la symbologique associée à la couche
self.iface.legendInterface().refreshLayerSymbology(zLayer)
#changer la visibilité : oui / non
if QGISVersionID > 10400 :
      self.iface.legendInterface().setLayerVisible(zLayer, nCond)
```

11.14 - Manipuler les couleurs

Dans la programmation Python (Qt), l'encodage des couleurs utilise la fonction « **Qcolor(r, g, b)** ». Les exemples ci-dessous montrent comment inverser la couleur RGB (pour obtenir BGR par croisement des canaux B et R) ou comment passer d'une couleur RGB en valeur HexaDécimale.

Exemple de code :

```
def InvRGB(zColor):
    zColor = long(zColor)
    zInvColor = 0
    if zColor<0 :
      Red = 0
      Green = 0
      Blue = 0
    else:
      Red =   int(zColor % 256)
      Green = int(zColor // 256 % 256)
      Blue =  int(zColor // 256 // 256 % 256 )
    zInvColor = QColor(Blue, Green, Red)
    return zInvColor

def rgb_to_hex(zColor):
    zColor = long(zColor)
    zInvColor = 0
    if zColor<0 :
      r = 0
      g = 0
```

```
    b = 0
  else:
    r = int(zColor % 256)
    g = int(zColor // 256 % 256)
    b = int(zColor // 256 // 256 % 256 )

  rgb = (b, g, r)
  hexColor = '#%02x%02x%02x' % rgb
  return hexColor
```

11.15 - Tester si un fichier ne contient d'objets

QGIS ne supporte pas en entrée des fichiers sans objets. Une solution consiste à tester le contenu du fichier en amont d'un chargment avec la librairie **GDAL / OGR**.

Exemple de code :

```
try:
   from osgeo import gdal
   from osgeo import osr
   from osgeo.gdalconst import *
except:
   import gdal
   import osr

#deux exemples de drivers
driver = ogr.GetDriverByName("MapInfo File")
driver = ogr.GetDriverByName("Shapefile")

datasource = driver.Open(uLayer[cLayer])
layer = datasource.GetLayer()
zFeatureCount = layer.GetFeatureCount()

        if zFeatureCount !=0:
                vLayer = self.iface.addVectorLayer(uLayer[cLayer],cLayer,"ogr")
```

11.16 - Créer des tuiles d'images avec GDAL sur un service WMS

L'exemple ci-dessous présente un exemple de la démarche. On crée un fichier XML « **zdestXML** » en passant des paramètres (« **zURL** », « **zSRS** », ...) connus ou récupérés, puis on charge cette ressource raster « **self.iface.addRasterLayer** ».

Exemple de code :

```
        f = file(zdestXML, "w")
```

```
        f.write("<GDAL_WMS>")
        f.write("    <Service name=\"WMS\">")
        f.write("        <Version>"+zVersion+"</Version>")
        f.write("        <ServerUrl>"+zURL+"</ServerUrl>")
        f.write("        <SRS>"+zSRS+"</SRS>")
        f.write("        <ImageFormat>"+zFormatImg+"</ImageFormat>")
        f.write("        <Layers>"+zLayersWMS+"</Layers>")
        f.write("        <Styles>"+zStyles+"</Styles>")
        f.write("    </Service>")
        f.write("    <DataWindow>")
        f.write("        <UpperLeftX>"+str(zBBOX0)+"</UpperLeftX>")
        f.write("        <UpperLeftY>"+str(zBBOX1)+"</UpperLeftY>")
        f.write("        <LowerRightX>"+str(zBBOX2)+"</LowerRightX>")
        f.write("        <LowerRightY>"+str(zBBOX3)+"</LowerRightY>")
        f.write("        <SizeX>"+str(zSizeW* 2 ^ 25 )+"</SizeX>")
        f.write("        <SizeY>"+str(zSizeH* 2 ^ 25 )+"</SizeY>")
        #f.write("        <SizeX>"+str(zSizeW)+"</SizeX>")
        #f.write("        <SizeY>"+str(zSizeH)+"</SizeY>")
        #f.write("        <SizeX>268435456</SizeX>")
        #f.write("        <SizeY>134217728</SizeY>")
        f.write("    </DataWindow>")
        f.write("    <Projection>"+zSRS+"</Projection>")
        f.write("    <BlockSizeX>1024</BlockSizeX>")
        f.write("    <BlockSizeY>1024</BlockSizeY>")
        f.write("    <BandsCount>3</BandsCount>")
        f.write("    <Cache>")
        f.write("        <Path>./gdalwmscache</Path>")
        f.write("        <Depth>2</Depth>")
        f.write("        <Extension>.jpg</Extension>")
        f.write("    </Cache>")
        f.write("    <MaxConnections>2</MaxConnections>")
        f.write("    <Timeout>300</Timeout>")
        f.write("    <OfflineMode>false</OfflineMode>")
        f.write("    <AdviseRead>false</AdviseRead>")
        f.write("    <ClampRequests>false</ClampRequests>")
        f.write("</GDAL_WMS>")
        f.close()

        znamefile = os.path.basename(zdestXML)
        vLayer = self.iface.addRasterLayer(zdestXML, znamefile)
```

NB : le temps de traitement peut demander plusieurs minutes. Les tuiles sont souvent de qualité médiocre si l'emprise est trop importante. Il n'y a pas de rafraichissement des jeux d'images en fonction des actions dans l'interface QGIS (zoom, déplacement).

11.17 - Créer une instance de composition (mise en page)

L'exemple ci-dessous montre comment lancer une nouvelle mise en page avec un bloc carte « **composerView.addComposerMap** » :

Exemple de code :

```
def MakeComposer(self):
    self.canvas = self.iface.mapCanvas()
    mapRenderer = self.canvas.mapRenderer()
    self.iface.actionPrintComposer().activate(0)
    composerList = self.iface.activeComposers()
    if(len(composerList) < 1):
        return

    composerView = composerList[0]

    c = composerView.composition()
    if(c is None):
        return

    c.setPlotStyle(QgsComposition.Preview)

    marge = 20.0
    dpi = c.printResolution()
    dpmm = dpi / 25.4
    width = int(dpmm * c.paperWidth())
    height = int(dpmm * c.paperHeight())

    x, y = 10, 10
    w, h = c.paperWidth()- marge, c.paperHeight()- marge
    composerMap = QgsComposerMap(c,x,y,w,h)
    composerView.addComposerMap(composerMap)
    composerView.setComposition(c)
    composerView.setPaintingEnabled(False)

    mView.addComposerMap(composerMap)
    mView.setComposition(c)
    mView.setPaintingEnabled(False)
```

11.18 - Récupérer la symbologie à défaut par type de géométrie

Exemple de code :

```
#par rapport à la géométrie d'une couche
zSymbol = QgsSymbolV2.defaultSymbol(zLayer.geometryType())

#Pour chaque type de géométrie
zSymbol = QgsSymbolV2.defaultSymbol(QGis.Point)
zSymbol = QgsSymbolV2.defaultSymbol(QGis.Line)
zSymbol = QgsSymbolV2.defaultSymbol(QGis.Polygon)
```

11.19 - Récupérer la symbologie à défaut d'une couche

Exemple de code :

```
#ancienne symbologie
symbols = zLayer.renderer().symbols()
symbol = symbols[0]

#nouvelle symbologie
symbols = zLayer.rendererV2().symbols()
symbol = symbols[0]
```

11.20 - Récupérer la version de QGIS

Exemple de code :

```
QGISVersionID = 0
try:
   QGISVersionID = int(unicode( QGis.QGIS_VERSION_INT ))
except:
   QGISVersionID = int(unicode( QGis.qgisVersion )[ 0 ])
```


Ressources, territoires, habitats et logement
Énergie et climat
Développement durable
Prévention des risques
Infrastructures, transports et mer
Présent
pour
l'avenir