



Formation

Module II



Formation Python pour QGIS 3

2022

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

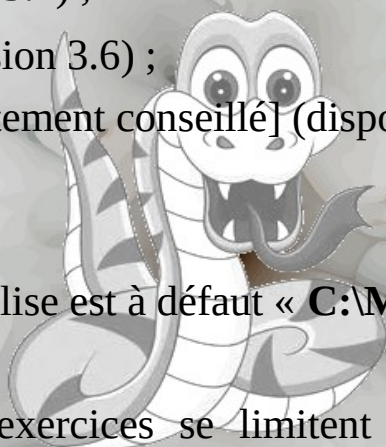
MINISTÈRE
DE LA COHÉSION
DES TERRITOIRES
www.cohesion-territoires.gouv.fr



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 ...



Python, un langage orienté objet ...

Python, un langage orienté objet Les CLASSES ...

Concepts



Python, un langage orienté objet ...

Python, un langage orienté objet Les CLASSES ...

```
import threading
import time
import tkinter as Tk
import math
```

Mais non !!!!!

classe permettant de gérer un Timer

Plutôt ça.

```
class MyTimer:
    def __init__(self, tempo, target, args= [], kwargs={}):
        self._target = target
        self._args = args
        self._kwargs = kwargs
        self._tempo = tempo

    def _run(self):
        self._timer = threading.Timer(self._tempo, self._run)
        self._timer.start()
        self._target(*self._args, **self._kwargs)

    def start(self):
        self._timer = threading.Timer(self._tempo, self._run)
        self._timer.start()
```



Python, un langage orienté objet ...

Concepts

L'introspection :

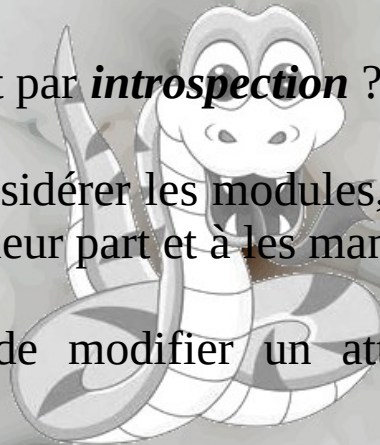
Grâce à un usage intensif des dictionnaires (conteneur associatif développé avec des tables de hachage), Python permet d'explorer les divers objets du langage (**introspection**) et dans certains cas de les modifier (**intercession**).

Mais qu'entend-on exactement par **introspection** ?

L'**introspection** consiste à considérer les modules, les fonctions comme des objets, à obtenir des informations de leur part et à les manipuler !

Il est possible de lire ou de modifier un attribut dynamiquement avec les fonctions :

- **hasattr**(objet, 'nom_attribut')
- **getattr**(objet, 'nom_attribut')
- **setattr**(objet, 'nom_attribut', nouvel_attribut)



Python, un langage orienté objet ...

Concepts

L'introspection :

- **type** :

- Exemple :

- type**(objet) >> renvoie le type de l'objet

- **dir** :

- Exemple :

- dir**(objet) >> renvoie toutes les méthodes, fonctions supportées par l'objet.

- **help** :

- Exemple :

- help**(objet) >> renvoie l'aide de l'objet.

- **__dict__**, **__doc__** :

- Exemple :

- objet.__dict__** >> renvoie le dictionnaire interne des attributs de la classe d'objet.





Python, un langage orienté objet ...

Exercice :

- Depuis la **console Python de QGIS**, à l'aide d'une boucle et de la fonction **getattr**, introspecter le premier niveau de l'objet « **qgis** » ;

```
objetQgis = dir(qgis)
print(objetQgis)

for mydir in objetQgis:
    print(str(mydir.upper())+' : ' + str(dir(getattr(qgis, mydir))))
```

Exercices



Python, un langage orienté objet ...

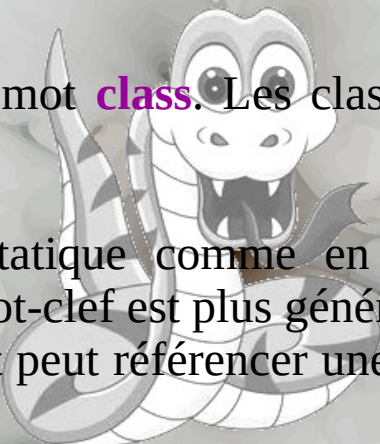
Concepts

La programmation objet est très bien supportée par Python : tous les types de base, les fonctions, les instances de classes (les objets « classiques » des langages C++ et Java) et les classes elles-mêmes (qui sont des instances de méta-classes) sont des objets.

Une classe se définit avec le mot **class**. Les classes Python supportent l'héritage multiple.

Il n'y a pas de surcharge statique comme en C++, mais le mécanisme des arguments optionnels et par mot-clef est plus général et plus flexible.

En Python, l'attribut d'un objet peut référencer une variable d'instance ou de classe (le plus souvent une fonction).



Python, un langage orienté objet ...

Concepts

Python fournit un mécanisme élégant et orienté objet pour définir un ensemble pré-défini d'opérateurs : tout objet Python peut se voir doté de fonctions dites spéciales.

Ces fonctions, commençant et finissant par deux tirets de soulignement (underscores), sont appelées lors de l'utilisation d'un opérateur sur l'objet :

- `+` (fonction `__add__`),
- `+=` (fonction `__iadd__`),
- `[]` (fonction `__getitem__`),
- `()` (fonction `__call__`),

... Des fonctions comme `__repr__` et `__str__`, permettent de définir la représentation d'un objet dans l'interpréteur interactif et son rendu avec le mot clé `print`.

Les possibilités sont nombreuses et sont décrites dans la documentation du langage.

La fonction spéciale d'initialisation d'une classe est incontournable :

`def __init__(self, arg1, arg2, ...):`

L'incontournable constructeur d'une classe



Python, un langage orienté objet ...

Concepts

Un petit exemple pour nous éclairer !!!

```
# Ecrire une classe Vecteur
class Vecteur :
    def __init__(self,abs,ord) :
        self.abs=abs
        self.ord=ord

    def __add__(self,autre) :
        return Vecteur(self.abs+autre.abs,self.ord+autre.ord)

    def __repr__(self) :
        return 'MonVecteurSomme('+str(self.abs)+' '+str(self.ord)+')
```

```
4 >>> v1 = Vecteur(2,3)
5 >>> v2 = Vecteur(20,30)
6 >>> v = v1 + v2
7 >>> v
8 MonVecteurSomme(22,33)
```

Après avoir tapé 'v'
On obtient le résultat 'MonVecteurSomme(22,33)'



Python, un langage orienté objet ...

```
class Personne:
```

```
    def __init__(self, nom, prenom):
```

```
        self.nom = nom
```

```
        self.prenom = prenom
```

```
    def identite(self):
```

```
        return self.nom.upper() + " " + self.prenom
```

```
class Etudiant(Personne):
```

```
    def __init__(self, niveau, nom, prenom):
```

```
        Personne.__init__(self, nom, prenom)
```

```
        self.niveau = niveau
```

```
    def presenter(self):
```

```
        return "Je m'appelle " + Personne.identite(self) + ", Etude : " + self.niveau
```

```
#Exemple d'appel :
```

```
e = Etudiant("Licence INFO", "Dupontel", "Albert")
```

```
print(e.presenter())
```

```
Je m'appelle DUPONTEL Albert, Etude : Licence INFO
```

Procédure d'initialisation
d'une classe
(instance)

Fonction
supportée par la classe

La classe Etudiant hérite
de la classe Personne



Python, un langage orienté objet ...

```
class Personne:
```

```
    def __init__(self, nom, prenom):
```

```
        self.nom = nom
```

```
        self.prenom = prenom
```

```
    def presenter(self) :
```

```
        return self.nom.upper() + " " + self.prenom
```

```
class Etudiant(Personne):
```

```
    def __init__(self, niveau, nom, prenom):
```

```
        Personne.__init__(self, nom, prenom)
```

```
        self.niveau = niveau
```

```
    def presenter(self):
```

```
        return "Je m'appelle " + Personne.presenter(self) + ", Etude : " + self.niveau
```



Attention à l'utilisation des méthodes en surcouches :

```
print(Personne("Dupontel", "Albert").presenter())
```

DUPONTEL Albert

```
print(Etudiant("Licence INFO", "Dupontel", "Albert").presenter())
```

Je m'appelle DUPONTEL Albert, Etude : Licence INFO





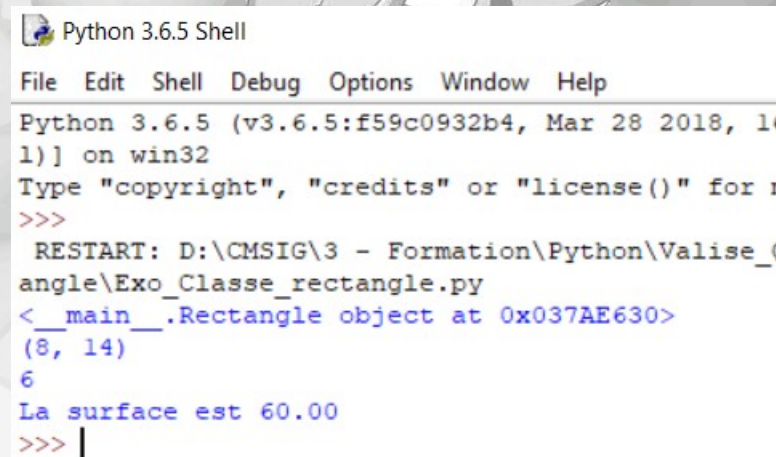
Python, un langage orienté objet ...

Exercices

Créer une classe « **Rectangle** » avec les fonctions :

- d'initialisation ;
- d'obtention du point bas-droit ;
- d'obtention de la largeur du rectangle ;
- de la surface du rectangle.

On fera un print de l'instance de classe (objet) et de chaque méthode pour visualiser les informations retournées.



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16
1)] on win32
Type "copyright", "credits" or "license()" for m
>>>
RESTART: D:\CMSIG\3 - Formation\Python\Valise_C
angle\Exo_Classe_rectangle.py
<__main__.Rectangle object at 0x037AE630>
(8, 14)
6
La surface est 60.00
>>> |
```





Python, un langage orienté objet ...

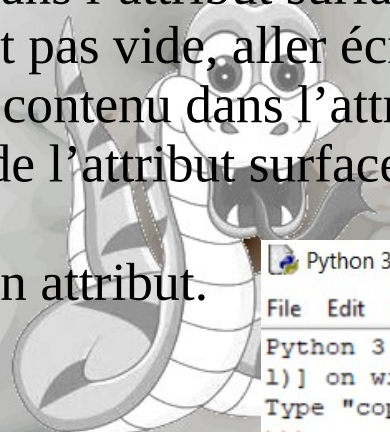
Exercices

Créer une classe «**TableauNoir**» avec les fonctions :

- d'initialisation pour manipuler un attribut appelé « surface »;
- d'écrire un message dans l'attribut surface ;
Si ce dernier n'est pas vide, aller écrire à la ligne
- d'afficher le message contenu dans l'attribut surface;
- d'effacer le message de l'attribut surface.

Donc, trois méthodes, et un attribut.

..... à vos claviers.



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.19
1)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: D:\CMSIG\3 - Formation\Python\Valise_Q3_2019\Exercices\
_rectangle_tableau_noir.py

première lecture
Coucou, comment vous sentez-vous à ce moment de la formation ?

seconde lecture
Coucou, comment vous sentez-vous à ce moment de la formation ?
J'attends pas forcément de réponse

Message effacé
le message est effacé
>>> |
```



Python, un langage puissant ...

Quelques **exemples** d'instructions sur les modules (import, ajout, chemins ...) :

```
import sys
```

```
sys.path #retourne les répertoires système (Python) de recherche de modules.
```

Exemple de retour dans la console :

```
['C:/user/PythonPourQGIS', 'C:\\Users\\user\\.qgis\\Python\\plugins\\openwor', 'C:\\Python32\\Lib\\idlelib', 'C:\\WINDOWS\\SYSTEM32\\python32.zip', 'C:\\Python32\\DLLs', 'C:\\Python32\\lib', 'C:\\Python32', 'C:\\Python32\\lib\\site-packages']
```

```
sys.__dict__ #Retourne toutes les informations d'un module, d'une classe
```

Pour changer les répertoires de recherches de modules :

```
sys.path['C :\\rep_1\\rep_2'] #doublez l'antislash
```

Pour ajouter un ou des répertoires de recherche :

```
sys.path.append('C:\\rep_1\\rep_2') #doublez l'antislash
```



Python, un langage puissant ...

Voici 2 façons de faire.

os.walk(Path) et **glob.glob(Path)**

La fonction `os.walk(path)` crée un générateur de triplets (`root`, `dirs`, `files`) dans l'arborescence de `path`. Un triplet est généré par répertoire visité. `root` représente le chemin d'accès du répertoire visité. `dirs` est la liste des sous-répertoires du répertoire `root` et `files` est la liste des fichiers du répertoire `root`.

```
import glob
import os.path

def listdirectory(path):
    fichier=[]
    l = glob.glob(path+"\\*")
    for i in l:
        if os.path.isdir(i): fichier.extend(listdirectory(i))
        else: fichier.append(i)
    return fichier
```

```
import os.path

def listdirectory2(path):
    fichier=[]
    for root, dirs, files in os.walk(path):
        for i in files:
            fichier.append(os.path.join(root, i))
    return fichier
```

```
import time

def compare(path):
    a = time.clock()
    listdirectory(path)
    b = time.clock()
    listdirectory2(path)
    c = time.clock()
    return b-a, c-b

print(compare('c:/MOC_STAGE'))
```



Python, un langage puissant ...

Les fonctions « built-in »

L'interpréteur Python dispose de fonctions embarquées immédiatement utilisables :

Built-in Functions				
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

Nous en avons déjà utilisé certaines !

Source documentaire : <https://docs.python.org/2/library/functions.html>



Python, un langage puissant ...

Concepts

Les fonctions « built-in »

La fonction **open** permet d'ouvrir un fichier texte ou binaire en lecture ou en écriture :

```
#pour ouvrir un fichier en lecture  
fichier = open("c:\\temp\\fichier.txt", "r")
```

```
#pour ouvrir un fichier en écriture  
fichier = open("c:\\temp\\fichier.txt", "w")
```

```
#un bloc d'instructions  
...
```

```
#pour fermer un fichier  
fichier.close()
```

les méthodes usuelles sur un objet file :

#écrire dans un fichier

write()

#lire tout le contenu d'un fichier

read()

#lire ligne par ligne

readline()

#lire tout le contenu d'un fichier, chaque ligne marquée d'un « \n »

readlines()

Sources documentaires :

<https://docs.python.org/2/library/stdtypes.html#builtin-file-objects>

<http://www.pythonforbeginners.com/files/reading-and-writing-files-in-python>





Python, un langage puissant ...

A vous de jouer :

Créer un fichier « system.txt » des commandes suivantes :

```
import sys  
sys.path #retourne les répertoires système (Python) de recherche de modules.
```

Le fichier « system.txt » devra contenir un répertoire par ligne comme le montre la figure ci-dessous

```
system.txt - Bloc-notes  
Fichier Edition Format Affichage ?  
D:\CMSIG\3 - Formation\Python\Valise_Q3_2019\Exercices\Did_Cmsig_formation_octobre_2013  
C:\Python365\python36.zip  
C:\Python365\DLLs  
C:\Python365\lib  
C:\Python365  
C:\Python365\lib\site-packages  
D:\CMSIG\1 - National
```

..... à vos claviers.

Exercices



Python, Gérer les erreurs (robustesse) ...

try :

"try" indique que la suite du programme sera surveillée par une gestion d'exception, quelle soit laissée par défaut ou qu'une nouvelle gestion soit implémentée.

except :

"except" permet la gestion d'une ou plusieurs exceptions. Le mot réservé "except" peut-être suivi de plusieurs arguments :

except: prend toutes les exceptions quelles que soit leurs types.

except <nom>: prend en compte l'exception <nom>.

except (<nom1>, <nom2>): Intercepte n'importe laquelle des exceptions listées.

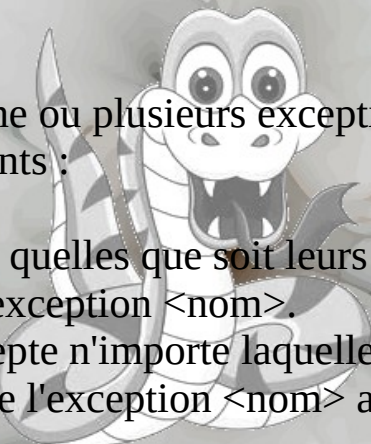
except <nom>, <valeur>: intercepte l'exception <nom> avec sa donnée <valeur>.

else :

"else" permet d'exécuter la suite du programme au cas ou aucune exception ne serait survenue.

finally :

"finally" applique toujours le bloc, quoi qu'il se passe.



Python, Gérer les erreurs (robustesse) ...

Il est aussi possible en Python de lever des exceptions, ceci se fait par le mot réservé **raise**.

L'instruction **assert** permet de rajouter des contrôles pour le débogage d'un programme.

Pour plus d'information sur la gestion des exceptions et l'arbre d'exceptions, reportez-vous à la documentation en ligne.

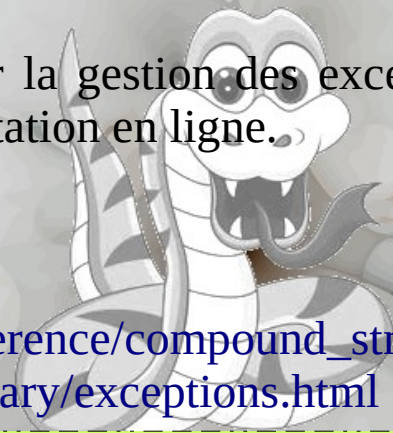
Pages d'information :

http://docs.python.org/2/reference/compound_stmts.html#try

<http://docs.python.org/2/library/exceptions.html>

```
try:
    import Tkinter
except ImportError:
    print("Le module Tkinter est nécessaire pour ce programme.\nImpossible de continuer.")

try:
    import wxPython
except ImportError:
    print("Le module wxPython est nécessaire pour ce programme.\nImpossible de continuer.")
```



Python, de multiples librairies et outils ...

Concepts

La bibliothèque standard de **Python** est très riche et de nombreuses bibliothèques gratuites peuvent être facilement utilisées pour un développement. La liste ci-dessous n'est pas exhaustive :

Bibliothèques scientifiques :

Calcul : NumPy, SciPy, PyIMSL Studio, SymPy, SAGE, ...

Système expert : pyCLIPS, pyswip, ...

Visualisation : pydot, matplotlib, pyngl, MayaVi, ...

Datamining : Orange, ...

Simulation : simPy, ...

...

Framework Web : Django, Karrigell, ...

Cartographie : TileCache, FeatureServer, Cartoweb 4, Shapely, GeoDjango, ...

Vecteur : Geojson, OWSLib, Quadtree, Rtree, Shapely, WorldMill, ...

ORM : SQLAlchemy, Storm, Django ORM ...

Driver SGBD : PyGresQL, Psycopg, MySQL-python ...



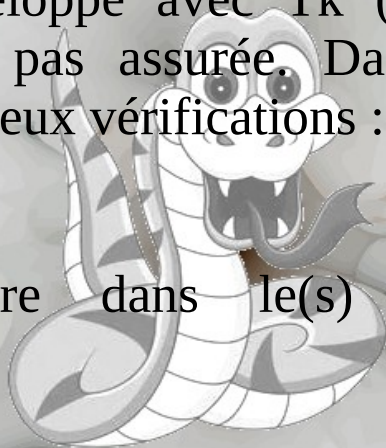
Python, créer des interfaces graphiques (ui) ...

Concepts

Il existe plusieurs librairies dédiées aux interfaces graphiques sous Python, souvent dépendantes du système d'exploitation.

Ainsi, si **IDLE** a été développé avec Tk (**Tkinter**), sa présence dans la distribution Windows n'est pas assurée. Dans tous les cas, la réalisation d'interfaces va donc induire deux vérifications :

- Disposer de la librairie ;
- Importer cette dernière dans le(s) module(s) « constructeur(s) » d'interface(s).



Python, créer des interfaces graphiques (ui) ...

Concepts

Pour créer les interfaces graphiques, il existe aussi des logiciels dédiés, qui peuvent être utiles, mais ces logiciels restent souvent liés à des bibliothèques (Gtk, Qt, wxwidgets...) :

Exemples :

- Pygtk :
- Tkinter :
- wxPython :
- Qt Creator
- Glade



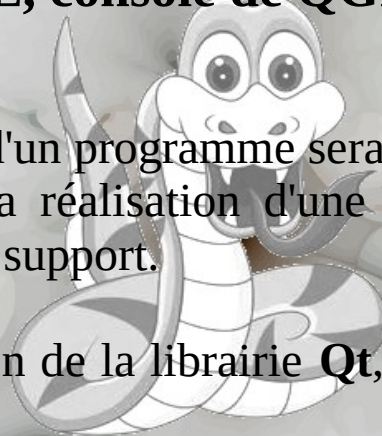
Python, créer des interfaces graphiques (ui) ...

Concepts

Pour un développeur confirmé, la solution la plus rapide restera de décrire la classe correspondant à son interface directement depuis son éditeur « **Python** » (**IDLE, console de QGIS, NotePad, PsPad ...**).

La question des interfaces d'un programme sera détaillée lors de la présentation des grandes étapes pour la réalisation d'une extension pour **QGIS** dans le module III et IV du présent support.

Notamment avec l'utilisation de la librairie **Qt**, embarquée dans la distribution du logiciel SIG.



Python, allez plus loin ...

Concepts

La classe OS (fichiers, répertoires ...) :

Cette classe fournit une manière portable d'utiliser les fonctionnalités du système d'exploitation dépendant.

Si vous souhaitez simplement lire ou écrire un fichier, voir **open()**, si vous voulez manipuler les chemins, regardez le module **os.path**, et si vous voulez lire toutes les lignes de tous les fichiers sur la ligne de commande voir le module **fileinput**.

Pour créer les fichiers temporaires et les répertoires voir le module **tempfile** et de haut niveau fichier et répertoire de manutention voir le module **shutil**.



Extensions spécifiques à un système d'exploitation particulier sont également disponibles via le module os, mais leur utilisation est bien sûr une menace pour la portabilité.



Python, allez plus loin ...

Concepts

La classe OS (fichiers, répertoires ...) :

La classe **os** comporte aussi la fonction **walk()** pour un parcours de répertoires et de fichiers.

Un exemple :

```
import os
for dirname, dirnames, filenames in os.walk(path):
    for subdirname in dirnames: print("Dossier [" + dirname + "] sous-dossier : " + subdirname)
    for filename in filenames: print("Fichier : " + str(os.path.join(dirname,filename)))
```

Une autre classe est disponible pour des opérations sur les fichiers. Le module **glob** trouve tous les chemins d'accès correspondant à un motif spécifié selon les règles utilisées par le shell Unix.

Sources documentaires :

<http://docs.python.org/2/library/os.html>

<http://docs.python.org/2/library/glob.html>





Python, allez plus loin ...

Exercices

Créer une fonction qui va permettre de lister tous les fichiers d'un certain type (« txt », « wor », « shp », « tab », « odt » ...) d'un dossier ou d'un disque à l'aide des fonctions de la classe **os**.

La fonction sera intitulée « **listdirectory** » .

Pour la tester, on analysera le contenu du répertoire racine de la valise « **Python pour QGIS** » (normalement, « **c:\MOC_STAGE** »).



Un peu d'aide ???



`os.path.join` => concatène les éléments dans la parenthèse
`os.path.splitext` => éclate la chaîne d'un path complet (nom et extension)



Python, allez plus loin ...

Concepts

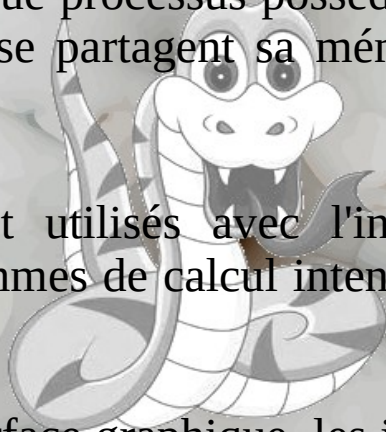
Le threading (encapsulation et contrôle de processus) :

Un **thread** ou fil (d'exécution) ou tâche (terme et définition normalisés par ISO/IEC 2382-7:2000), est similaire à un processus car tous deux représentent l'exécution d'un ensemble d'instructions du langage machine d'un processeur. On parle de parallélisation mais, là où chaque processus possède sa propre mémoire virtuelle, les threads d'un même processus se partagent sa mémoire virtuelle (mais pile d'appel propre).

Les threads sont typiquement utilisés avec l'interface graphique ("GUI") d'un programme ou par des programmes de calcul intensif (par exemple l'encodage d'une vidéo).

En effet, dans le cas d'une interface graphique, les interactions de l'utilisateur avec le processus, par l'intermédiaire des périphériques d'entrée, sont gérées par un thread, tandis que les calculs lourds (en termes de temps de calcul) sont gérés par un ou plusieurs autres threads. Cette technique de conception de logiciel est avantageuse dans ce cas, car l'utilisateur peut continuer d'interagir avec le programme même lorsque celui-ci est en train d'exécuter une tâche.

Pour les calculs intensifs, l'utilisation de plusieurs threads permet de paralléliser les traitements (sur machines multiprocesseur => rapidité).



De python à Qgis ...

Concepts

QGIS utilise largement les ressources de la librairie graphique **QT**. Connaître a minima cette dernière vous permettra de développer des interfaces très conviviales ...

Présentation de la librairie graphique de **DIGIA** :

Qt (prononcé officiellement en anglais cute (/kju:t/) mais couramment prononcé Q.T.) :

C'est une API orientée objet et développée en C++ par **Qt Development Frameworks**, filiale de la société **DIGIA**.

Qt offre des composants d'interface graphique (widgets), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution, d'analyse XML, etc.

Par certains aspects, elle ressemble à un framework lorsqu'on l'utilise pour concevoir des interfaces graphiques ou que l'on architecture son application en utilisant les mécanismes des signaux et slots par exemple.



De python à Qgis ...

Concepts

QGIS utilise largement les ressources de la librairie graphique **QT**. Connaître a minima cette dernière vous permettra de développer des interfaces très conviviales ...

PyQt, c'est quoi ?

PyQt est un module libre qui permet de lier le langage Python avec la bibliothèque **Qt** distribué sous deux licences : une commerciale et la **GNU GPL**. Il permet ainsi de créer des interfaces graphiques en Python.

Et un Binding, c'est quoi ?

Et bien **PyQt** est un binding reliant Python à **Qt**, mais aussi :



PySide est un binding qui permet de lier le langage Python avec la bibliothèque **Qt**, disponible à l'origine en C++. **PySide** se distingue de **PyQt**, le binding historique, par le fait qu'il est disponible sous licence LGPL. Pour conclure, **PySide** est un projet initié et soutenu jusque fin 2011 par **NOKIA**.

Pour information, **Qt** supporte des bindings avec plus d'une dizaine de langages autres que le C++, comme Java, Python, Ruby, Ada, C#, Pascal, Perl, Common Lisp, etc.



De python à Qgis ... (Pont)

Trois ponts





Formation Python pour QGIS 3



Fin du module

Merci de votre attention !!!!

