Python pour les administrateurs systèmes et réseaux

Jean-Marc Pouchoulon
Rectorat de Montpellier

Pour le groupe *Laser Languedoc Roussillon*

Python

- Python est un langage objet de très haut niveau , dynamiquement typé.(Tout objet est manipulable à l'exécution).
- Python est guidé par quelques grands principes qui s'appliquent partout.
- Python est multi-plateforme et produit du byte-code.
- Python est donné avec un ensemble conséquent de modules prêts à l'emploi.
- Python s'interface bien avec du C.

Zen de Python

- Au départ Python est fait pour le prototypage rapide:On développe, on identifie les goulots d'étranglements, on développe en C ensuite les fonctions les plus lentes et on les mappe en objet Python
- Dans les faits il y a déjà beaucoup de travail de fait et les modules sont déjà optimisés.
- Python dispose d'une console interactive très pratique
- import this (pour voir le zen de Python)

Ma Comparaison avec d'autres languages

- Plus simple que Perl mais moins de choix en termes de modules. Python n'est pas fait pour le one liner.
- Vis à vis de Java il est beaucoup moins verbeux. Par contre son modèle objet est moins académique (design patterns #).
  Il est aussi plus lent.
- Ruby est le grand concurrent.
- Php est à mon sens moins lisible.
- Plusieurs implémentations de python existent: cpython,jython,ironpython,boo,python for .net..

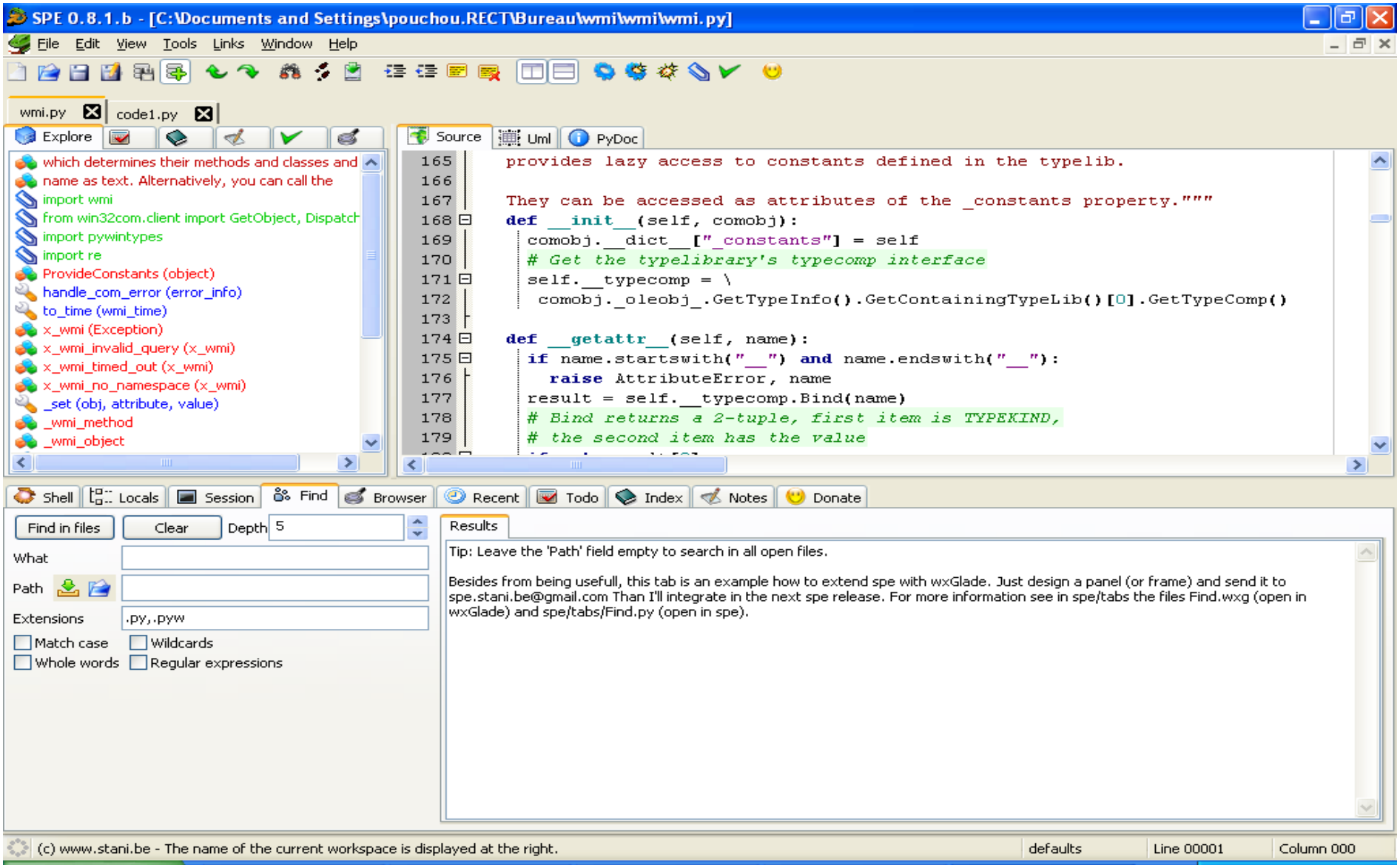
Python dans le monde « réel »

- Zope est la "killer" application de python.
- Python-ldap permet d'accéder et de gérer un annuaire LDAP.
- Twisted se pose comme une référence originale dans le domaine des réseaux.
- Python est un langage de prédilection pour XML.
- Python XPCOM permet de manipuler la suite Mozilla.
- Pythonwin32 permet de manipuler les objets com tels qu'excel word ...
- Python est inclus dans openoffice...
- RedHat développe ses assistants en python.
- L'institut Pasteur forme ses chercheurs à Python.
- Bit torrent..
- Google

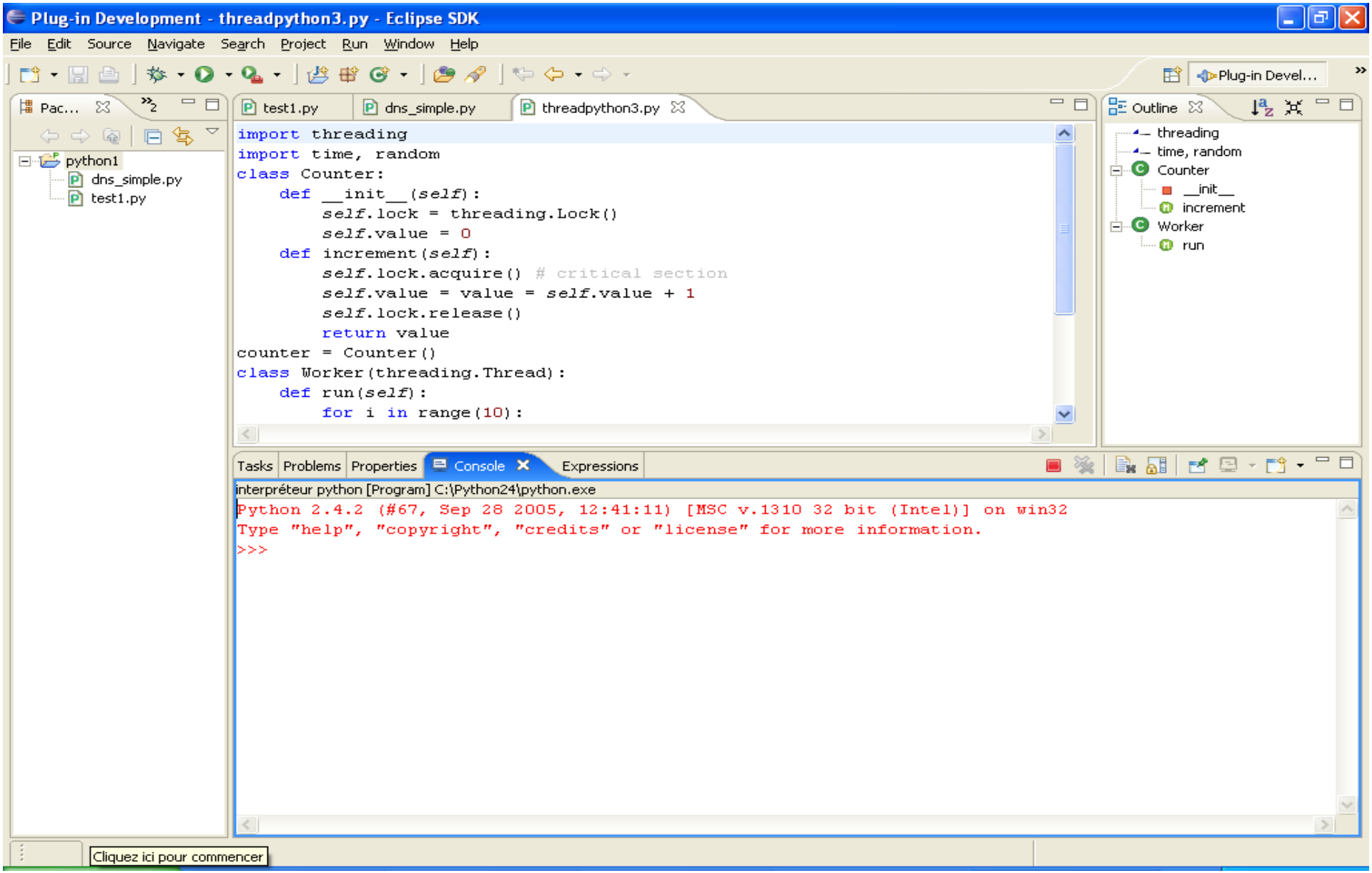
Objectifs

- L'objet de cette formation est de vous donner les éléments pour utiliser python dans un contexte d'administration systèmes et réseaux (par pour faire du développement d'applications graphiques) et de créer un groupe d'utilisateurs python pour échanger.

Editeurs: spe



Editeurs:Eclipse: Pydev



Préparation

- Chargement sur Python.org de python et du module win 32 et/ou image vmware.
- Premier pas avec pythonwin et/ou ipython
- Faite print "hello world"
- Rappeler la commande
- Completion, affichage des attributs
- Regarder la documentation

Bases du language

- L'interpréteur python est accessible en tapant python.
- On peut aussi créer un fichier (extension .py) avec shebang #!/usr/bin/python
- Sous windows de nombreuses consoles existent. Nous utiliserons pythonwin pour cette introduction. (il existe aussi pydev pour eclipse,jedit,spe...)
- ATTENTION : Python est sensible à la casse.
- Le code doit être **indenté.** (très clair , formateur et donc pas de parenthèses)

La gueule de Python

```
import monModule # Chargement d'un module

def  mafonction:
        """doc string qui permet de documenter"""
        instructions
        if condition:
                on fait ca
        else:
                on fait ca

a = mafonction(arguments)
```

Code Python

- Indentation : 4 car pas de TAB
- 1 Instruction par ligne.
- Pas plus de 80 car / Ligne
- Conventions :
  - Variables = minuscules
  - Classe = MixedCase
  - Méthodes = mixedCase
  - Constante = CONSTANTE

- voir PEP 8(Python Enhanced Proposal)

Obtenir de l'aide

- Comment avoir de l'aide:
  help(objet) ex `help(list)`
  `dir(list)` pour avoir les méthodes de l'objet (introspection toujours possible en Python)

- L'attribut `__doc__` contient un string de documentation en ligne.
- Il peut être formater via un simple print.

```
>>> print sys.exit.__doc__
  exit([status])...
```

Typage dynamique lors de l'affectation

```
>> a = 5
>> b = 6     # integer
>> a = 'zozo'  # string
>> a = "le petit 'bouchon'"
>> a
   "le petit 'bouchon'"
```

Trois """ permettent d'inhiber l'interprétation des caractères spéciaux.

```
>>> """ ceci ne bouge pas ``\^^ $1"""
   ' ceci ne bouge pas ``\\^^ $1'
```

Les variables

On peut affecter plusieurs variables en seule passe:

```
>>> a,b,c = 1,2, 'toto'

a,b,c = 1,2,'toto'
```

Les entiers

```
>>> a = 5
>>> type(a)
    <type 'int'>

>>> import sys  #  import d'un
                #  module on verra
                 # ensuite ce que c'est

>>> sys.maxint  # taille maximale d'un
                # entier
2147483647
```

Longs integers
Floating point numbers

```
>>> 200L
  200L
```

La seule limite de ce type est la mémoire de la machine.

```
>>> 200.37
  200.37
>>> 300e-2
  3.0
```

Les imaginaires

```
>>> a=(4+6j)
>>> a.real
4.0
>>> a.imag
6.0
>>>
>>> a = 4 + 5j
>>> b = 6 + 4j
>>> a +  b
(10+9j)
```

Fonctions sur les types numériques

- Les fonctions classiques abs(X), round(X),int,long(X), sont disponibles pour ces types.
- Les assignements de ce type +=, -= sont supportés.

```
>>> a = 3
>>> a += 1
>>> a
4
>>>
```

Le module operators

```
>>> a= 5
>>> b= 6
>>> from operator import *
>>> div(a,b)
0
>>> mul(a,b)
30
>>> floordiv(a,b)
0
>>> add(a,b)
11
```

Les Booléens

```
>>> a = 5
>>> b = 6
>>> a > 5
False
>>> a < b
True
>>> a = False
>>> b = True
>>> a or b
True
>>> a and b
False
>>> not ( 0 > 2 )
True
>>> a,b,c = True,False,True
>>> b=0
>>> a & b
0
```

AND et OR

Attention: 'and' et 'or' ne retournent pas True ou False.

- **`X and Y and Z`**
  - Si tous sont vrais, retourne la valeur de Z.
  - Sinon , retourne la première valeur fausse de la sous-expression.
- **`X or Y or Z`**
  - Si tout est faux, retourne la valeur de Z.
  - Sinon , retourne la première valeur vrai de la sous-expression.

- Astuce: Comment implémenter (test ? expr1 : expr2) en Python:

  resultat = test and expr1 or expr2

- Si test is True, resultat = expr1.
  - Si test est False, resultat = expr2.

Attention:
expr1 ne doit pas être Faux

Les Strings

On peut saisir de longue chaînes en python sur plusieurs lignes grâce à \.

```
>>> """ ceci est un long texte \
... bla bla `\
... fin """
' ceci est un long texte bla bla `fin '
```

Python concatène automatiquement les strings séparés par un espace.

```
>>> 'a' "toto" "5"
'atoto5'
```

Opérations sur les strings

Ajout:

```
>>> 'complet' + `5`
  'complet5'

>>> 'complet' + str(5)
  'complet5'
```

Multiplier les strings:

```
>>> a = "coucou
>>> a * 3
  'coucoucoucoucoucou'
```

Opérations sur les strings

```
>>> a = "Le mot le plus long"
>>> len(a)  # donne la longueur du string.
19
>>> a = "coucou"
>>> a.upper()
'COUCOU'
>>> b = "que d'or que d'or"
>>> b.find("or")
6
>>> b.find("or",7)  # On limite la
                    # recherche sur b[7:] voir le slicing.
15
```

Format de données

Comme en C on retrouve une syntaxe proche de printf

```
>>> "Il est %d heures , %s" % (11,'jean-marc')
'Il est 11 heures , jean-marc'
```

Ou encore à l'aide d'un tuple (voir plus loin) :

```
>>> f = (11,'jean-marc')
>>> "Il est %d heures , %s" % f
'Il est 11 heures , jean-marc'
```

Slicing

a	z	e	r	t	y	
0	1	2	3	4	5	index caractère
-5	-4	-3	-2	-1	0	index car inverse
0	1	2	3	4	5	6 index slice
-6	-5	-4	-3	-2	-1	0 index slice inverse

```
>>> z[1:]
'zerty'
>>> z[:4]
'azer '
 >>> z[-2]
't'
>>> z[-1]
'y'
>>> z[:-1]
'azert'
```

```
>>> z[::-2]
'yrz'
>>> z[::-1]
'ytreza'
```

Objets, identité, Types mutables et valeurs

- *'Objects are Python's abstraction for data. All data in a Python program is represented by objects or by relations between objects . **Every object has an identity, a type and a value**'*
- Un type mutable va supporter la modification de sa valeur.
- Un type non mutable ne pourra qu'être recréé si on veut le changer.(ex string)
- Pour voir l'identité d'un objet = **id(objet)**
- Pour voir le type de l'objet = **type(objet)**
- Notions qui vont s'éclairer par la suite.

Exo 1

- *Créer un string chaine 'azerty'*
- *Retrouvez son id (via la commande id(objet))*
- *Que se passe t il si on modifie le premier caractère (via un slicing) ?*
- *Comment le modifier ?*

Les types complexes: Listes

Les listes sont mutables. Il s'agit d'une collection d'objets, indexée comme les strings donc "slicibles". Les tuple sont proches des listes mais sont non mutables.

```
 >>> L =
[1,3,'a','test',['abc',345]]
>>> L
[1, 3, 'a', 'test', ['abc', 345]]
>>> L[1]
3
```

Listes

Peut contenir d'autres listes

```
>>> L =[1,3,'a','test',['abc',345]]
```

Slicibles

```
>>> L[1:3]
[3, 'a']
>>> L[4]
['abc', 345]
```

Isolation d'un sous éléments

```
>>> L[4][1]
345
```

Inversion

```
>>> L[:-1]
[1, 3, 'a', 'test']
>>>
```

Listes dynamiques

On peut générer des listes à partir d'expressions valides en python:

```
>>> [ x for x in range(1,10)]
[1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> [ x * x for x in range(1,10)]
[1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> [ x * y for x in range(1,10) for y
in range(1,4)]
[1, 2, 3, 2, 4, 6, 3, 6, 9, 4, 8, 12, 5,
10, 15, 6, 12, 18, 7, 14, 21, 8, 16, 24,
9, 18, 27]
```

Les types complexes: Tuples

- Idem que les listes mais non mutables
- Intérêt objets optimisés pour la lecture.
- Même opération que sur les listes.

```
>>> x = () # tuple vide
>>> x = 3,4,'toto' # pas besoin de parenthèses pour créer un tuple
>>> x
(3, 4, 'toto')
>>> x = (3,4,'toto') # mais ca marche aussi
>>> tuple('toto')
('t', 'o', 't', 'o')
>>> x[1:] # Le slicing fonctionne aussi
    (4, 'toto')
>>> x[1:]
>>> x[1] = 'p' # type non mutable au contraire des listes.
Traceback (most recent call last):
  ....
```

Types complexes: dictionnaires

- Type mutable équivalent au hash de Perl
- Ensemble de clef:valeur

```
>>> D1 = { 'Clement': 'grand',
           'Arnaud':'moyen' ,
           'Mathilde':'petite'}
```

Pour être une clef d'un dict un objet doit être hashable vérifiable via la fonction hash

```
>>> hash('toto')
  1380683436
```

Instructions de contrôle de flow

```
if (clause):
    instructions...
elif(clause):
    instructions...
else:
    instructions...
```

```
while(condition):
    instructions...
else:
    instructions exécutés à la
    fin de la boucle.
```

```
For  variables in sequence:
    instructions....
```

Exo 2

- *Créer un dictionnaire via dict()*
- *Ajouter des valeurs à l'aide D[clef]=valeur (prendre par exemple les os (windows , linux, mac , vms , aix , solaris)comme clefs et vos appréciations perso comme valeur.*
- *Retrouver une valeur avec D.get(clef)*
- *Lister les clefs/valeurs à l'aide de la fonction items() via une boucle for.*
- *Lister les clefs via keys() et les valeurs via values()*

Exo 3

- *Créer une liste des jours de la semaine*
- *Copier là lCopie = [:]*
- *Enlever samedi et dimanche via la function pop*
- *Les rajouter via append*
- *Extraire les éléments jeudi,vendredi de la liste.(slicing)*
- *Afficher la liste inversée via un slicing[::-1]*
- *Faire un tri de la liste via sort*
- *Lister un éléments sur 2 de la liste avec un tri décroissant*
- *Quel est la différence entre l.reverse()*
  et l[::-1] ?
- *Faire un cast de la liste en string avec un séparateur « ; »*

Pour cela utiliser la syntaxe ";".join(liste)

Les Références

Comme en JAVA on crée une référence sur un objet, un changement sur l'objet est visible sur l'ensemble des références.
On peut créer des références circulaires.

```
>>> a = [1,2,3,4,5]
    >>> b = a
    >>> b
    [1, 2, 3, 4, 5]
    >>> a[1]=6
```

Attention

```
>>> a= list('azerty')
>>> a
['a', 'z', 'e', 'r', 't', 'y']
>>> b=a
>>> b
['a', 'z', 'e', 'r', 't', 'y']
>>> del a
>>> b
['a', 'z', 'e', 'r', 't', 'y']
>>>
```

Références

```
>>> a
[1, 6, 3, 4, 5]
>>> b
[1, 6, 3, 4, 5]
>>> id(a) # La fonction id donne l'identifiant d'un objet
14556336
>>> id(b) # On vérifie qu'ils sont identiques.
14556336
>>>
```

Copie d'objets

- Listes:

```
 >>> a
[1, 6, 3, 4,
5]
>>> c = a[:]
>>> c
[1, 6, 3, 4,
5]
>>> a[1]=2
>>> a
[1, 2, 3, 4,
5]
>>> c
[1, 6, 3, 4,
5]
```

- Dicos:

```
>>> D1 =
{'Mathilde':
'petite', 'Clement':
'grand', 'Arnaud':
'moyen'}
>>> D2 = D1.copy()
>>> D2
{'Mathilde':
'petite', 'Clement':
'grand', 'Arnaud':
'moyen'}
>>>
```

Exo 4:shallow et deep copy

- *Créez une copie L2 de L1*

```
>>> L1 = [['un','deux','trois'],1,2,3]
```

Vérifiez via la commande "is" que L1 est égal à L2

Enlevez 'un' de L1.
Qu'y a t il dans L2 ?
Utilisez le module deepcopy et refaites les mêmes tests.

Que veux donc dire shallow et deepcopy ?

Les fonctions en python.

```
def addition(arg1,arg2,argn...):
        instructions 1
        intructions 2
        ... n
        return(objet1, objet2,objn...)
```

Les arguments peuvent être n'importe quels objets. Ces objets sont passés par référence
Une fonction même sans return retourne None... (équivalent de null)

Exemple de fonction

```
def prouvemod(a,b):
    a = 1
    b = 2


c = 5
d = 6
prouvemod(c,d)
print "c = %d, d = %d" %(c,d)
>>> c = 5, d = 6
```

Fonction: arguments

Une fonction peut recevoir des **arguments nommés :**

```
def fonction(a=5,b='toto'):
connect (host='www.ac-montpellier.fr',
  port=8080)
```

Fonctions:Nombre variables d'arguments

Quand on ne connait pas le nombre d'arguments , nommés ou non on peut définir une fonction de la manière suivante pour lui faire passer un nombre variables d'arguments.

```
def fonction(*arguments,**dictionnaire):

        for arg in arguments:
                print arg
        for k,j in dictionnaire.items():
                print k,j

args = (3,4,5)

dico ={ 'prenom':'jeanmarc','nom':'pouchoulon'
   }
#Appel de la fonction
fonction(*args,**dico)
```

Exo 5: Type mutables et fonctions

- *Créer une fonction qui modifie une liste qui lui est passée en argument. (utiliser range(n) pour générer une liste de n nombre et dans la fonction affecter le premier élément de la liste à la fonction).*
- *Affichez la liste après passage à la fonction.*

La liste a t elle été modifiée ?

- *Essayer de remplacer la liste par une liste différente du même nom dans la fonction.*
- *Afficher la liste d'origine.*

Apply

# apply est équivalent à la forme précédente
apply(fonction,argument)

```
def addition(*arguments):
    somme = 0
    print arguments
    for arg in arguments:
        somme = somme + arg
    return(somme)


print addition(1,2,3,4,5)

print apply(addition,range(1,10))

>>>
(1, 2, 3, 4, 5)
15
(1, 2, 3, 4, 5, 6, 7, 8, 9)
45
```

Map

```
#Liste = map(fonction,sequence)


L=[1,3,-6,-8]
T=(1,3,-6,-8)
resultT = map(abs,T)
print resultT
resultL = map(abs,L)



>>> [1, 3, 6, 8]
>>> [1, 3, 6, 8]
```

Filter

filter(fonction,sequence)

Si fonction(item) renvoie vrai on sélectionne l'item.
Si la séquence est un string ou un tupple elle retourne un string ou un tupple sinon une liste

```
import string
mots = string.split(open(r"H:\toutpython\formation    python\modules.txt",'r').read())


def au_moins_10(mots) :
    return len(mots) >= 10

mots_long = filter(au_moins_10, mots)
for mot in mots_long: # retourne les mots de plus de 10 caractères
    print mot
```

Reduce

reduce(fonction,sequence)

on applique une fonction à une sequence

```
def add(x,y): return x+y

y = (2,3,5)
print reduce(add,y)
print reduce(add,range(2,2000))


>>10
>>1998999
```

Lambda

```
#lambda argument: opération.
a = lambda _: _+1


>>> a(1)
2
>>> a(2)
3
>>>
```

le _ signifie valeur de la dernière instruction évaluée.

NB : Filter , map , reduce sont fortement « chahutées » par le fondateur de Python et dans la communauté Python et pourrait disparaître avec python 3000

Espace de nom

- Un espace de nom est un mappage entre des variables et leurs valeurs. (via un dictionnaire)
- 2 variables peuvent donc avoir le même nom mais dans des espaces de noms différents.
- Un objet python accède aux espaces de nom au travers des scopes (ou encore portées) local, global et builtins.

Espace de nom: example les fonctions

```
def f1(x,y):
        a =1
        b =2
        print locals() # accès au scope local
        print globals() # accès au scope global
                        # et builtin.
>>> f1('a','b')
{'a': 1, 'y': 'b', 'b': 2, 'x': 'a'} # variable locale.
```

Lecture-écriture de fichiers

```
from pprint import pprint # package pretty
                          # printer améliore
                          # l'affichage

f = open(r"C:\fich2.txt",'w')
f.write('coucou')
print >> f,'\nune autre facon de faire'
print >> f,"\n".join(['faire un retour a la ligne'])
f.close()
```

Relecture de fichier

```
f = open(r"C:\fich2.txt") # ouverture en lecture par
                          # défault
lines = f.readlines()


for ligne in lines:
    if  len(ligne) > 1:
        print ligne
print f.name # nom du fichier
print f.mode # mode d'ouverture
f.seek(0) # retour au début du fichier.
for ligne in lines: # elimine les lignes vides
    if  len(ligne) > 1:
        print ligne
```

ipython en 5 mn

- Ipython est un shell python. Il permet de mixer les commandes shell et Python
  voir http://ipython.scipy.org/.
  la complétion(tab), le rappel des commandes, et la sauvegarde des sessions fonctionnent par défaut
- lsmagic permet de voir les commandes shell accessibles
- history permet d'avoir le rappel des commandes.
- edit 4:7 permet d'éditer les lignes 4 à 7
- _i2 permet de rappeler la ligne 2, _2 permet de rappeler l'output de la ligne 2.
- ?nomobjet permet d'avoir de l'aide sur un objet.
- %sc monString = ls -al (shell capture met le resultat de la capture dans monString)
- %sc -l (ou %sx) idem que précédent mais le résultat est stocké sous forme de liste.
- !ls lance la commande ls
- $variable python permet d'utiliser la variable python dans une commande shell

ipython example

```
In [30]:%sc mesFichiers = ls

In [31]:for fichier in mesFichiers.l:
   ....:     !ls -al $fichier
   ....:
-rw-r--r--  1 root root 256260787 oct  3 08:59
  access1.log
-rwxr-xr-x  1 root root 235 déc 14 15:05 concat.py
-rwxr-x---  1 root root 452 fév  5  2005
  ctrlFRESHCLAM
...
```

Module Python

Un module python est en fait un fichier qui va contenir du code python. Le fait d'utiliser la commande import module permet d'accéder aux objets de ce modules via **module.nom .**
Un module a un attribut __name__ qui est initialisé avec le nom du fichier.
Si on on fait un import du module __name__ va être égal à __main__ dans ce cas on sait que le fichier n'est pas importé mais lancé en ligne de commandes.
On peut importer un module ou un élément de ce module dans l'espace de nom dans lequel on travaille via la commande from. Un module a donc son propre espace de nom.

Module et espace de nom

```
>>> import time # On importe le module.
>>> time.time() # Pour appeler un élément du module on commence
1095347147.5439999 # par le nom du module suivi de la fonction
>>> time.ctime()
'Thu Sep 16 17:05:55 2004'
>>> from time import time # avec le from on importe la fonction
                         # dans notre espace
>>> time()                # de nom
1095347180.2309999

>>> from time import ctime
>>> ctime()
'Thu Sep 16 17:06:45 2004'
>>>
```

from module import *

Un from module import * peut poser des problèmes sous windows du fait que le system ne soit pas sensible à la casse. Le module va être rechercher en suivant sys.path.
On peut aussi importer un ensemble de sous modules contenus dans un **package**.
(Un **package** est donc un ensemble de modules avec une arborescence, il doit y avoir présent un fichier __init__.py , même vide dans chaque directory).

Globals() locals()

```
>>> def f1(x,y):
        a =1
        b =2
        print locals()  # accès au scope local
        print globals()  #  accès au scope global et
                            #builtin.

>>> f1('a','b')
{'a': 1, 'y': 'b', 'b': 2, 'x': 'a'} #  variable
                                       # locale.

{'f1': <function f1 at 0x00E1FBB0>, 'pywin':
<module .....
```

Principaux modules:sys

Sys
Permet d'intéragir avec le système:

argv - argument de la ligne de commande;
argv[0] nom du script
path - chemin de recherche des modules;
path[0] est la directory du script

modules - dictionnaire des modules chargés.

Le module sys:path

Le module sys permet d'indiquer à python le chemin des modules que l'on souhaite importer:

```
>>> import sys
>>> sys.path
['', 'C:\\WINDOWS\\System32\\python23.zip',
'C:\\Python23\\lib\\site-packages\\Pythonwin',
'C:\\Python23\\lib\\site-packages\\win32',
'C:\\Python23\\lib\\site-packages\\win32\\lib',
'C:\\Python23\\lib\\site-packages',
'C:\\Python23\\DLLs', 'C:\\Python23\\lib',
'C:\\Python23\\lib\\plat-win',
'C:\\Python23\\lib\\lib-tk', 'C:\\Python23']
```

On peut modifier le chemin via :

```
 sys.path.append(r"C:\Documents and
Settings\pouchou.RECT\Bureau\Docs diverses
(D)\formation"
```

EXO 5: module OS

- faire les imports nécessaires :

```
import os,fnmatch ; from time import ctime
```

- Quel le pid de pythonwin ?
- Quel est le current directory ?
- Lister la liste des variables d'environnement.
- Changer de directory
- Tester si C:\windows est une directory ou un fichier ?
- Obtenir la date de modification d'un fichier.
- Lister une liste de fichier *.py (os.listdir) dans C:\windows\python24\lib à l'aide de fnmatch.
- A l'aide de de os.walk lister l'arborescence de C:\python24\lib\site-packages.

Lancement de commandes:popen

```
>>> essais = os.popen('dir')
>>> for line in essais:
        print line
```

```
Le volume dans le lecteur H s'appelle Offline
Le numero de serie du volume est 0000-0000

    Repertoire de H:\toutpython\formation python

   10/09/2004  17:33    <REP>          .
```

Lancement de commande:popen2

- permet de contrôler stdin et stdout

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

import os
montexte = """
                ceci est un texte qui va servir pour un exemple
                de ma très modeste introduction à python
                association d'idées
                python = beau
                python = fort
                python = magnifique
                python = rapide

            """

w,r = os.popen2("grep '='")
w.write(montexte)
w.close()
print r.read()

~
~
~
~
~
"ospopen2.py" 19 lines, 396 characters
```

Lancer une commande: module commands

- Unix uniquement, simple

```
>>>
>>>
>>>
>>> import commands
>>> print commands.getoutput('ls  *.py')
child.py
montokenize.py
ospopen2.py
process1.py
process2.py
unicode.py
utf8.py
>>> print commands.getstatusoutput('cat *.poc')
(256, 'cat: *.poc: No such file or directory')
>>>
```

Lancer une commande: Module subprocess

- **Python 2.4 uniquement.**

```
>>> from subprocess import PIPE
>>> from subprocess import Popen
>>> cmd = "ls -al"
>>> pipe=Popen(cmd,close_fds=True,shell=True,stdout=PIPE).stdout
>>> pipe.readline()
'total 13804\n'
```

- **pipe renvoie un file descripteur que l'on peut lire après.**
- **Shell =True indique que la commande va être passer par l'interpréteur shell.**
- **close_fds fait que tous le file descriptor seront fermés à la mort du fils.**

Lancer un process: fork,exec

os.fork() créé un nouveau process en python., copie de l'original.

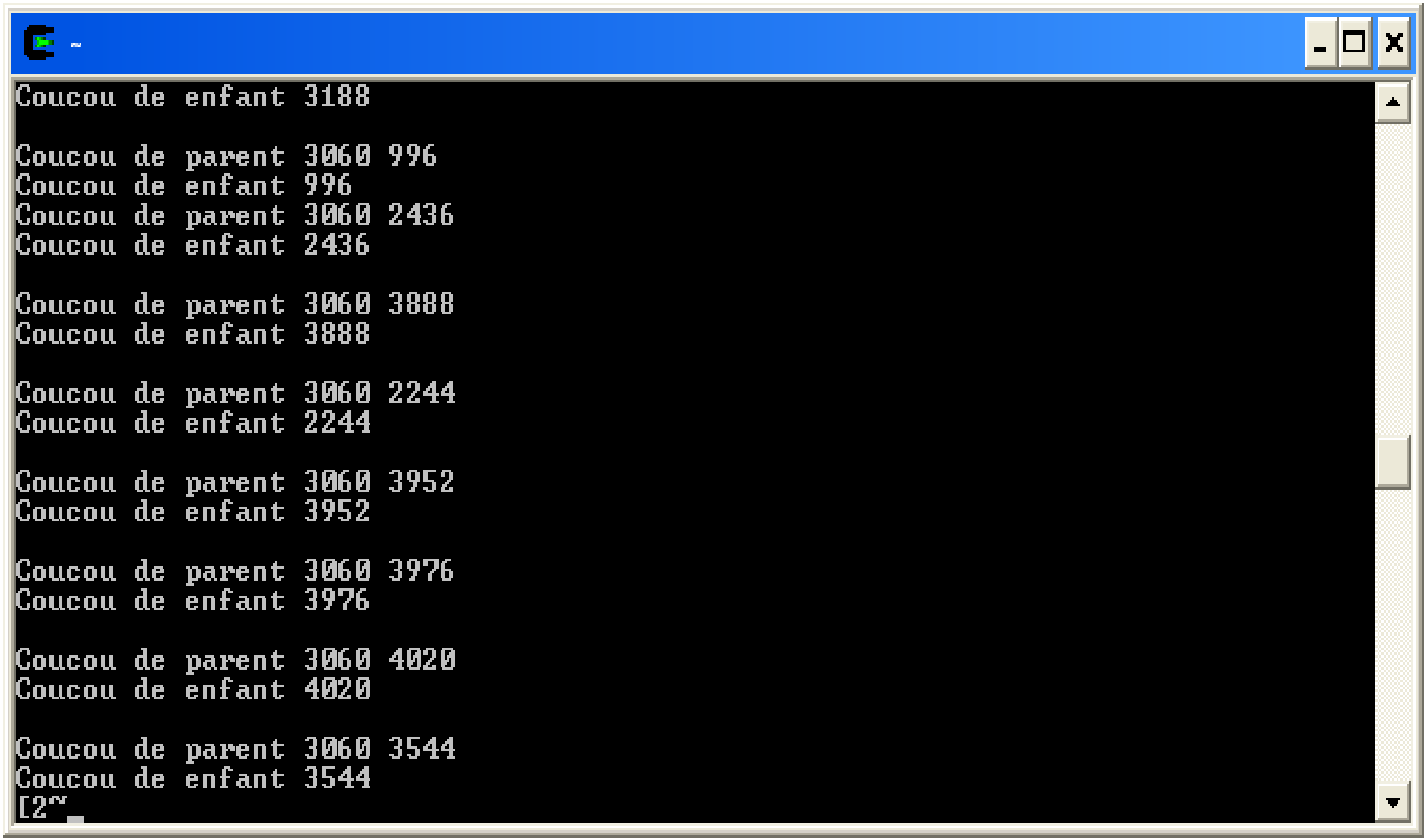
Dans le processus enfant os.fork retourne une valeur égale à zéro ce qui permet de le différencier du processus parent.

Exemple: Fork

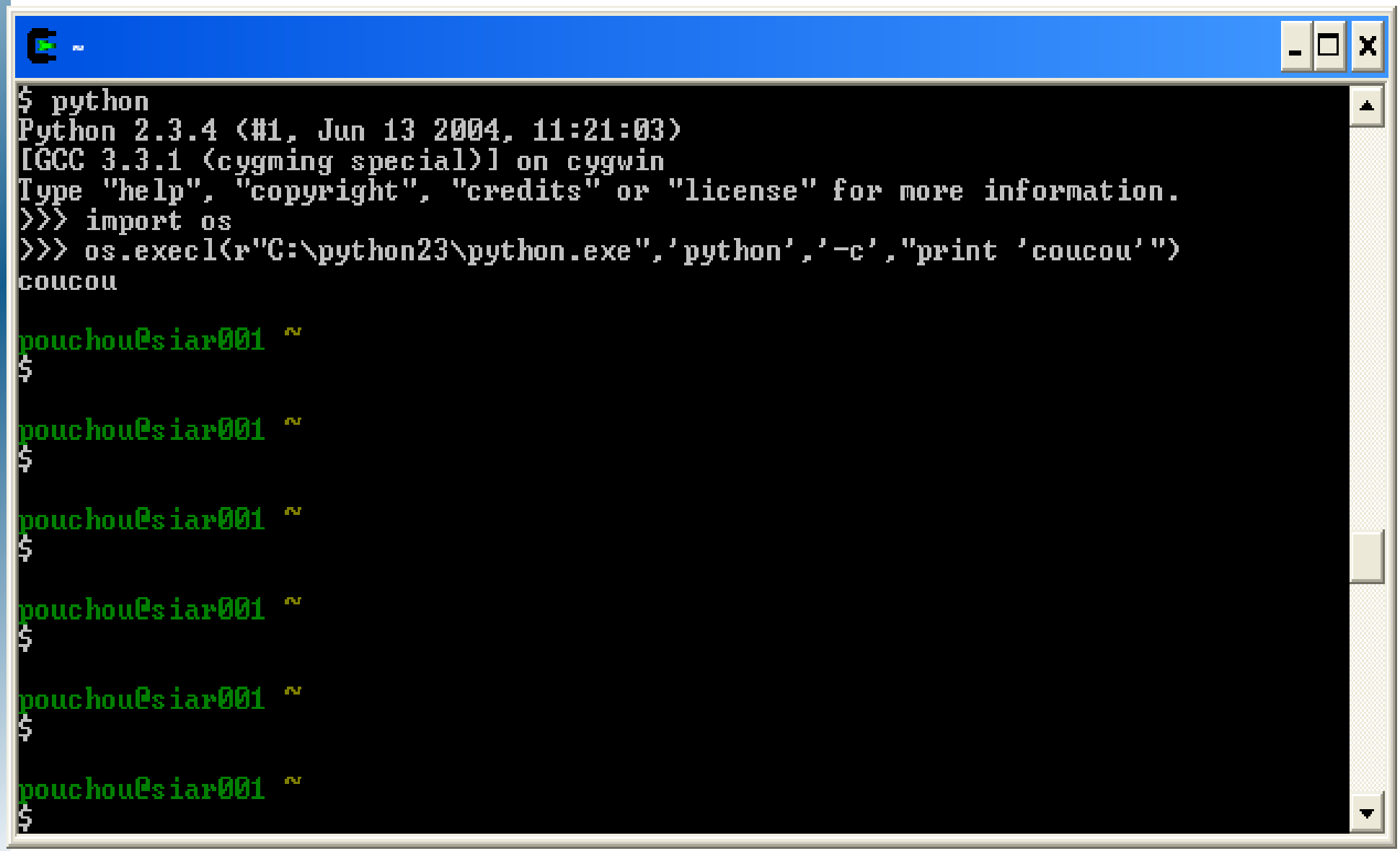
```
import os
def enfant():
    print 'Coucou de enfant',  os.getpid()
    os._exit(0)
def parent():
    while 1:
        nouveaupid = os.fork()
        if nouveaupid == 0:
            enfant()
        else:
            print 'Coucou de parent', os.getpid(), nouveaupid
        if raw_input() == 'q': break

parent()
```

Exemple fork: resultat



Exemple exec



Exo 6

- *Donner une liste des utilisateurs unix sur votre machine virtuelle vmware.*
- *Pour cela utiliser le module commands et la commande getent passwd et une list comprehension.*

Objets en Python:Définitions

Un objet Python a:

- Une **identité** (que l'on retrouve en appelant un id(objet) et qui est unique).
- Des **attributs** , (fonctions ou variables)
- Un **nom**. Les mappages entre des noms et leurs objets s'appelle un « **espace de noms** ».

Objet en Python:Définitions

- Les relations entre les objets sont de 2 ordres :

« *Est une sous-classe de* »: Par exemple un dauphin « est une sous-classe de mammifère ».

« *Est une instance de* »: Par exemple flipper « est une instance » de dauphin.

- Elles s'expriment avec **attributs spéciaux**
  - **type** = l'attribut __class__
  - **bases** (Une ou plusieurs) = attribut __bases___

Objets:ATTENTION !!

Tout est objet en python donc :

Une instance est un objet aussi (donc avec un type).

Un type est un objet aussi. (avec un type aussi). Un type est aussi appelé métaclasse c'est à dire une classe dont les instances sont des classes. (sert rarement directement).

'object' est le type de tous les types et dont le type est 'type'.

Squelette de classe

```
class nom_de_classe[(classes_de_bases)]:
"""docstring de ma classe"""

# variables de classe
var1 = valeur1
var2 = valeur2

def __init__(self,arguments):
        """Docstring initialiseur d'instance """
        #self signifie cette instance = this
        #en Java
        self.arg1  = arg1
        self.arg2  = arg2

def maMethode(self,argument):
        """ Docstring """
        # code de ma méthode
```

Exemple de classe

```
class Mammiferes(object):#classe object
      pass                          # Mère de toutes les classes
class AnimauxMarins(object):
      pass

class Dauphin(animauxmarins,mammiferes):
            # héritage multiples

>>> print dauphin.__class__
<type 'type'>
>>> print dauphin.__bases__
(<class '__main__.animauxmarins'>, <class
   '__main__.mammiferes'>)
# création d'une instance
flipper = Dauphin()
```

Constructeur de classe

```
class dauphin(animaux,mamiferes):
    def __init__(self,age):
        # super permet d'appeler l'initialiseur de la
        # classe parente  en
        # respectant la MRO
        self.age=age
        self.nom=nom
        super(dauphin,self).__init__()
```

Comportement identique pour une méthode quelconque. Si la méthode redéfinit une méthode de classe parente il faut utiliser super pour appeler la méthode parente

Attention:

`__init__` renvoie toujours None

Ancien et nouveau modèle de classe

- Il existe de types de modèles de classes en python, le deuxième existant depuis la version 2.2 de python. Depuis python 2.2 il n'y a plus de séparation entre les types built-in (list dictionnaire tuple...). Les classes de type classique sont de type **classobj** et peuvent s' écrire sous la forme class classique:

- ```
  >>> import types
  >>> class classic:
  ...     pass
  ...
  >>> types.ClassType is type(classic)
  True
  >>> type(types.ClassType)
  <type 'type'>
  ```

Le type type est aussi le type de ces classes.

Les deux types de classes peuvent coexister.
  ```

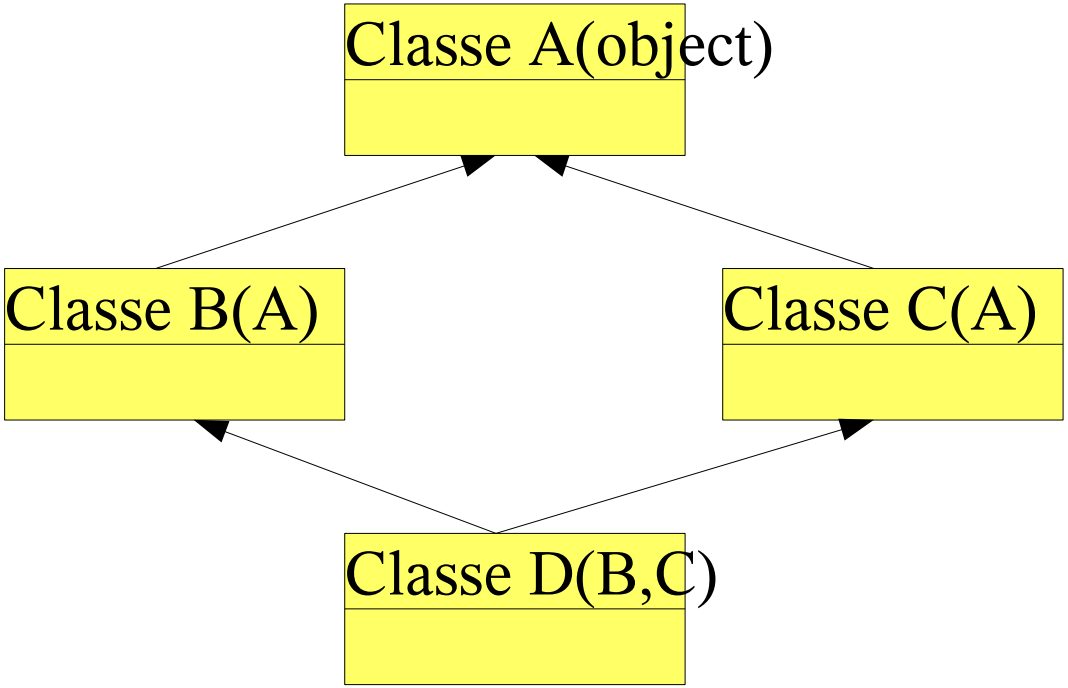
M.R.O ou ordre de résolution des références d'attribut

- Avec le nouveau modèle de classe , soit 4 classes A,B,C,D.

Tel que :

```
class A(object):
class B(A):
class C(A):
class D(B,C):
```

M.R.O ou ordre de résolution des références d'attribut



Dans le cas d'un type classique l'ordre de recherche va être d'abord en profondeur et ensuite sur la droite (soit DBACA) .

Le fait que la classe A apparaisse 2 fois est génant , c'est pour cela qu'avec les nouvelles classes l'ordre de résolution est devenu de droite à gauche puis en profondeur. (soit DBCA).

Méthodes spéciales

- Beaucoup de comportement peuvent être redéfinies:l'addition , la soustraction, le slice... via des méthodes spéciales précédées de __ ex:

```
>>> a = 'azerty'
>>> dir(a)
['__add__', '__class__', '__contains__',
  '__delattr__', '__doc__', '__eq__', '__ge__',
  '__getattribute__', '__getitem__',
  '__getnewargs__', '__getslice__', '__gt__',
  '__hash__', '__init__', '__le__', '__len__',
  '__lt__', '__mod__', '__mul__',....
```

Exo 6: Classe et méthode spéciales opérations sur les complexes

- *Créez une classe Complexes(object) et redéfinissez l'addition de 2 complexes via __add__*
- *Modifier l'affichage des instances d'objet via __repr__ (afficher partie réelle: partie imaginaire)*

Property

L'objet property permet d'avoir un attribut dynamique (getter/setter Java avec élégance)

```
class Multiplier(object):
        def __init__(self,x,y):
                self.x = x
                self.y = y
        def getMulti(self):
                return (self.x * self.y)
        def setMulti(self,_v):
                print "Pas affection de la valeur"
        def delMulti(self):
                print "Attribut Dynamique Pas de del"
        Multi=property(getMulti,setMulti,delMulti,"""test de l'objet property""")

m = Multiplier(3,4)
print m.Multi
m.Multi=5
del m.Multi

>>> 12
Pas affection de la valeur
Attribut Dynamique Pas de del
```

Notion de Persistence des objets.

La sauvegarde des objets ou sérialisation permet de garder l'état d'un objet entre 2 lancements d'applications.

Plusieurs façons de faire :

- Les modules **pickle/cPickle** permettent de sauvegarder un objet. (pickle ne sauvegarde pas le code de la classe).
- Les modules de type **anydbm** qui fournissent des bases de données sous la forme de dictionnaires stockés sur disque.
- Le module **shelve** synthèse des deux précédents.
- Utiliser la **zopedb** comme base de données objets.(module à installer en plus).

Exemple shelve

```
class test(object):
    def __init__(self,value):
        self.data = value
    def __repr__(self):
        return("dans test %s \n" % self.data)

t1 = test('valeur1')
t2 = test('valeur2')
t3 = test('valeur3')

monShelve = shelve.open(r'C:/monshelve7.shl','n')
monShelve['clef1'] = t1
monShelve['clef2'] = t2
monShelve['clef3'] = t3
print monShelve.items()
t1.data = "pas heure"
print t1
print monShelve.items()
monShelve.close()
```

Exemple shelve

Attention : Pas de modification de l'objet shelve par:

```
>>> [('clef2', dans test valeur2
  ), ('clef1', dans test valeur1
  ), ('clef3', dans test valeur3
  )]
  dans test pas heure

  [('clef2', dans test valeur2
  ), ('clef1', dans test valeur1
  ), ('clef3', dans test valeur3
  )]
```

Gestion des exceptions

Ex :La fonction float() permet de convertir n'importe quel type numérique en float:

- >>> float(123)
  123.0

- Une mauvaise saisie

```
>>> float('mauvais_input')
Traceback (most recent call last):
  File "<interactive input>", line 1, in ?
ValueError: invalid literal for float():
mauvais_input
```

Gestion des exceptions

```
def gestion_float(typenumerique):
        try:
                retval = float(typenumerique)
        except ValueError:
                retval = "mauvaise valeur"
        return retval
```

Dans ce cas on obtient :

```
>>> from gestion_float import gestion_float
>>> gestion_float("aaa")
  'mauvaise valeur'
>>> gestion_float(12334)
  12334.0
```

Squelette exceptions

```
try:
    #instructions...
except (e1,e2,en),libelle_erreur:
    #instructions "
finally:
    #instructions exécutées dans
    #tous les cas
```

TP1:Accéder à un annuaire ldap avec Python

- *Installer python-ldap sous linux/windows (sous linux python setup.py build;python setup.py install)*

- *Trouvez le nombre de mail commençant par « j » dans l'annuaire.*

nb : adresse de l'annuaire = x.ac-montpellier.fr port 389

Tp2: python et dns

A) Résolution de nom avec le package socket.

Après avoir importer le package socket :
1) quel est le nom dns de votre machine ?
2) Retrouver les infos sur le package et sur les fonctions relatives aux dns?
3) Quel est l'adresse de www.ac-montpellier.fr ?
4) Retrouvez son A record (son nom quoi) à partir de l'adresse ?
Retrouver la liste de tous les enregistrement de www.clamav.net noms de serveurs
Sur quel port le service telnet tourne –t-il sur la machine ?
Donner la représentation de l'adresse ip 'X.X.X.X' sur votre machine et revenez ensuite à l'adresse ip.

Python et dns: package dnspython

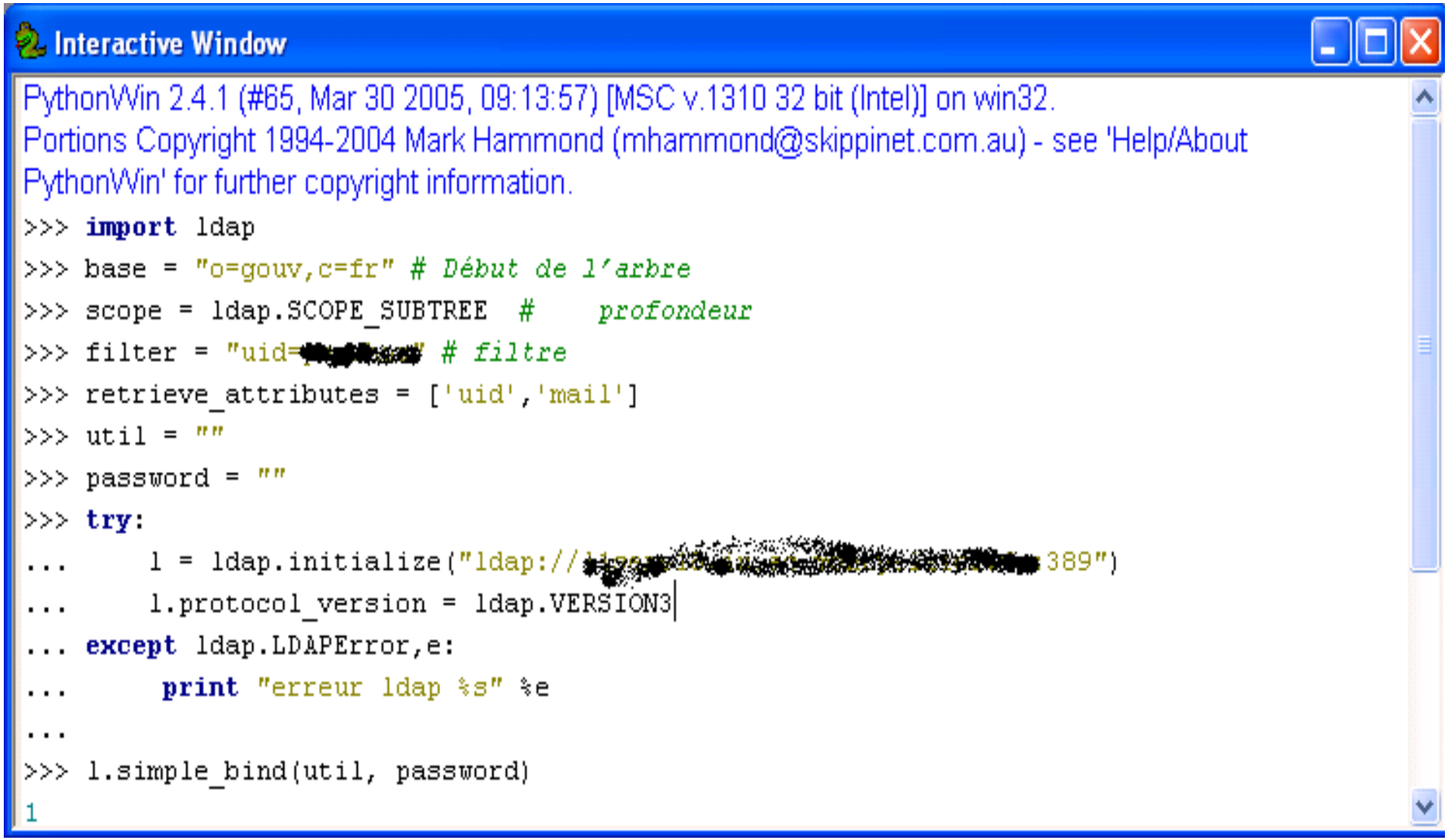
*B) **Utilisation du package pydns de http://www.dnspython.org/.***

Utiliser le module dns et dns.resolver.query(zone,type d'enregistrements) pour obtenir les MX du domaine ac-montpellier.fr.
La commande dig @serveur_de_nom AXFR zone permet d'obtenir l 'ensemble des enregistrements au dns.
Ex : dig @adrdns axfr ac-montpellier.fr.
Utiliser dns.zone.from_xfr(dns.query.xfr('adr ip serveur de nom', 'ac-montpellier.fr')) sur dns1 et dns2 afin de comparer les zones et vérifier que la réplication des zones entre les 2 dns fonctionnent bien.

ANNEXE 1 ldap

liens utiles:
http://python-ldap.sourceforge.net/
http://laser.igh.cnrs.fr/article.php3?id_article=40

Ouverture et bind de la connexion



Requête search synchrone

En mode synchrone le programme attend le résultat de la requête

```
>>> res = l.search_s(base,scope, filter, retrieve_attributes)
>>> for r in res:
...     print r
...
```

Ldap v3 rfc2251

L’enveloppe (PDU) contient le type de requête et un message ID

- LDAPMessage ::= SEQUENCE { messageID MessageID, protocolOp CHOICE

{ bindRequest BindRequest,
bindResponse BindResponse,
unbindRequest UnbindRequest,
searchRequest SearchRequest
….

Ldap v3 rfc2251

Result est le résultat de la requête
LDAPResult ::= SEQUENCE { resultCode ENUMERATED {
success (0),
operationsError (1),
protocolError (2),
timeLimitExceeded (3),
sizeLimitExceeded (4),
compareFalse (5),
compareTrue (6),
authMethodNotSupported (7),
strongAuthRequired (8),
referral (10),
 …..

Requête search asynchrone

```
ensembleDesResultats = [] # Liste que l'on va remplir des résultats
while 1:
        result_type, result_data = l.result(ldap_result_id, 0) # Recherche des résultats
        if (result_data == []): # si rien on repart dans la boucle
            break
        else: # si on a un resultat de type search alors on le rajoute à notre liste
            if result_type == ldap.RES_SEARCH_ENTRY:
                ensembleDesResultats.append(result_data)

for e in ensembleDesResultats:
    print e

'''
```

Requête modify (mode add)

```
try:
        l.modify_s(userDn,
        [
            (ldap.MOD_ADD,"objectclass",["posixaccount"]),
            (ldap.MOD_ADD,"objectclass",["shadowaccount"]),
            (ldap.MOD_ADD,"uidnumber", [`lastUidNumber`]),
            (ldap.MOD_ADD,"gidnumber", ["503"]),
            (ldap.MOD_ADD,"homeDirectory", [homeDir]),
            (ldap.MOD_ADD,"loginshell", ["/bin/bash"])
        ]
        )

 except ldap.SERVER_DOWN , error_message:
        print error_message
        sys.exit()
```

Utilisation d'une modlist

```
retrieve_attributes = []
dn, res = l.search_s(base,scope, filter, retrieve_attributes)[0]
dictModUser = copy.deepcopy(res) # copie du resultat de la requete
dictModUser["telephoneNumber"] ="04 67 93 49 73" # on modifie ce champ dans la copy

attrs = modlist.modifyModlist(res,dictModUser) # on construit une liste de modification
l.modify_s(dn,attrs) # On lance la requête de modification
```