IMAC - Python TD1

Prise en main du langage

Dans chaque paragraphe, il y a quelques lignes de code.
Copiez-les *une par une* dans l'interpréteur Python pour observer leur effet.
Essayez des variantes.
Ces exemples vont vous permettre de vous familiariser très vite avec le langage.
Rappel : on accède aux attributs d'un objet avec `dir(...)` et à l'aide en ligne avec `help(...)`
Après chaque paragraphe, il y a des petits exercices liés aux concepts présentés.

Tour d'horizon rapide

On lance l'interpréteur en tapant "python".
On obtient un prompt >>>.
On tape les instructions directement, il n'y a pas besoin de mettre un point virgule à la fin des lignes.

Entiers et chaînes

```
# ceci est un commentaire
print 2+2 # on affiche avec "print"
print "dro"+'madaire' # les chaînes peuvent être avec des " ou des ' et on peut les concaténer
print "a"+2*"na"+"s" # on peut aussi multiplier les chaînes
print "mignonne allons voir si la rose\netc." # on peut mettre des \quelquechose
print "ceci\test une tabulation, mais pas cela\\t." # le caractére d'échappement est \
alphabet="abcdefghijklmnopqrstuvwxyz" # on définit une variable
print alphabet # ça affiche l'alphabet, pas de surprise
print len(alphabet) # "len" c'est pour "length", c'est-à-dire "longueur"
alphabet.upper() # les chaînes ont des méthodes commodes
print (2**8)%100 # 2 puissance 8, le tout modulo 100 (ça fait 56)
print 1, # notez la virgule à la fin du print : elle permet de ne pas aller à la ligne
print 1,; print 2 # ... cette ligne affichera donc « 1 2 »
```

Exercice 1 Soit une variable appelée "message" et contenant un message de moins de 75 caractères. Écrire un bout de programme qui affiche ce message en l'encadrant avec des *.
Par exemple si message="coin coin", le programme doit afficher :

```
*************
* coin coin *
*************
```

Exercice 2 Soit toujours la même variable ; cette fois-ci on veut que le cadre fasse 79 caractères de large et que le message soit centré.
Par exemple (on a mis des points pour visualiser la mise en page, mais ils ne doivent pas apparaître, bien sûr) :

```
*******************************************************************************
* . . . . . .coin coin. . . . . *
*******************************************************************************
```

Le formatage des chaînes de caractères

Pour l'instant, nous avons utilisé des "+" pour concaténer les chaînes à afficher. Ce n'est pas la meilleure méthode.
Pour faire l'équivalent d'un `printf`, on fait comme ceci :

```
print "La somme de %d et %d, ça fait %d, qui est un nombre %s."%(2, 40, 2+40, "positif")
```

C'est-à-dire qu'on utilise l'opérateur modulo (%) qui va prendre à gauche, une chaîne avec des %quelquechose, et à droite, une liste d'arguments à remplacer.

Listes (ou tableaux, c'est pareil)

```
daltons=["joe","jack","william","averell"]
print daltons[0] # les listes, ça commence à zéro
print daltons[1:3] # sous-liste de l'élément 1 inclus jusqu'au 3 exclus
print daltons[-1] # -N = Nème élément en partant de la fin
print daltons[2:] # sous-liste de l'élément 2 jusqu'à la fin
daltons.sort() # tri alphabétique en place
print daltons # affiche la liste triée
petit=daltons[0] # c'est "averell" vu qu'on a trié dans l'ordre alphabétique
print petit[0] # surprise : une chaîne, c'est une liste de caractères...
print daltons[-2][3:] # on peut faire ça
```

Exercice 3 L'inverse de l'exercice 1 : on suppose qu'il existe une variable "message", contenant un message encadré d'étoiles.
Écrire un bout de programme permettant de retrouver le message original.
Par exemple, si message="*"*13+"\n* coin coin *\n"+"*"*13, le programme doit afficher "coin coin".
Bien sûr, il faut que ça marche quelle que soit la taille du message (pas seulement avec des messages de 9 caractères!).
On suppose que le message original est bien construit (c'est-à-dire qu'il n'y a pas d'étoiles en plus ou en moins que ce qu'il faut).

Dictionnaires

```
client={ "prenom":"lazare","nom":"garcin", "date_de_naissance":[1,4,1515],

"poissons_rouges":2, 123:"un deux trois"}

client["ville"]="plessis" # ajoute une clé
client["ville"]="paris" # modifie la valeur
encore={'tel':'888-88-888-88', 123:'cent-vingt-trois'} # nouvelles données
client.update(encore) # mise à jour
del client[123] # supprime une valeur
print client # affiche la table résultante
```

Les structures de contrôle

```
if 5>2: print "5 est plus grand que 2"
else: print "5 est plus petit que 2"

for nom in daltons: print nom+" est un dalton"


print "la famille dalton comprend :"
for nom in daltons:
    nom_complet=nom+"dalton"
    print nom_complet

# Plus élégant :

print 'La famille Dalton comprend :\n' + '\n'.join([x.capitalize()+ ' Dalton' for x in daltons])

# Cette dernière utilise une compréhension de liste, comme ce quicksort en deux lignes

def q(L):
    if len(L)<= 1: return L
    return q([x for x in L[1:] if x<L[0]])+[L[0]]+q([y for y in L[1:] if y>=L[0]])

s="abracadabrakangourou"
t=""
while s:
    if s[0]=="k": break
    t=t+s[0]
    s=s[1:]

print s
print t
```

On note qu'il n'y a pas d'accolade : c'est l'indentation qui détermine les blocs de code !

Exercice 4 Écrire un bout de programme qui prend deux chaînes s1 et s2 et qui les "tricote" ensemble, par exemple si s1="abcdef" et s2="0123456789", le résultat doit être "a0b1c2d3e4f56789" (on intercale une lettre de la première, une lettre de la deuxième, etc.)

Exercice 5 Écrire un bout de programme qui "détricote" une chaîne s en deux chaînes, l'une contenant tous les caractères de rang pair et l'autre ceux de rangs impairs.
Par exemple pour le "a0b1c2d3e4f56789" de la question précédente, cela donne "abcdef68" pour la première chaîne et "01234579" pour la deuxième.

Définition de fonctions

```
def fib(n):
    """Imprime les nombres de Fibonacci inferieurs a n"""
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b

fib(2000)

def demande_ok(prompt, essais=4, message="oui ou non, SVP!"):
    while True:
        ok = raw_input(prompt)
        if ok in ['o', 'oui', 'O', 'y', 'yes', 'ouaip']: return True
        if ok in ['n', 'non', 'N', 'no', 'nope']: return False
        essais-=1
        if essais < 0: raise IOError, 'Rien a faire ...'
        print message


def printf(format, *args):
    print format % args

printf('Le %s de mon %s ne mange du %s que le %s', 'cheval', 'cousin', 'foin', 'dimanche')

aa=[3,8]
*aa
range(*aa)

map(lambda x:x**2, range(10,20))
sum(_)


filter(lambda s:s.startswith('j'),daltons)
```

Exercice 6 On suppose qu'on a la liste suivante :

```
daltons=[{"nom":"joe","taille":140,"caractere":"teigneux"},

{"nom":"jack","taille":155,"caractere":"idiot"},
{"nom":"william","taille":170,"caractere":"stupide"},
{"nom":"averell","taille":185,"caractere":"abruti"}]
```

Ecrire une fonction `compare_taille` qui prenant en argument deux daltons (deux tables de hachage, en fait), comparant leur attribut "taille", et renvoyant :
- zéro s'ils sont de même taille
- un nombre négatif si le premier est plus petit
- un nombre positif si le premier est plus grand

Ecrire aussi une fonction `compare_nom` prenant en argument deux daltons, comparant leur attribut "nom", et renvoyant :
- zéro s'ils sont identiques
- un nombre négatif si le premier est *avant* le second dans l'ordre alphabétique
- un nombre positif si le premier est *après* le second dans l'ordre alphabétique

Exécuter le code suivant pour tester les fonctions :

```
print daltons
daltons.sort(compare_nom)
print daltons
daltons.sort(compare_taille)
print daltons
```

Interlude : comment faire un programme python?

Par convention, appeler le fichier `quelquechose.py` (ce n'est pas obligatoire, mais c'est comme appeler un source C `quelquechose.c`!)
Au début du source, mettre la ligne :
`#!/usr/bin/python` ou `#!/usr/bin/env python`
Puis rendre le fichier exécutable (`chmod +x toto.py`) .
On peut maintenant lancer le programme : `./toto.py arguments...`

Si les commentaires ou des chaînes du programme contiennent des caractèeres non-ascii (`ord(c)>127`), la première ligne du programme doit être

```
# -*- coding: utf-8 -*-
# ou
# -*- coding: iso-8859-15 -*-
# selon votre encodage
```

Quelques fonctions utiles

Si on veut transformer un entier en chaîne (on l'a fait plus haut), on utilise la fonction `str`. Inversement, pour transformer une chaîne en nombre, on utilise `int` ; par exemple : `int("2")+int("40")` ça fait `42`.

On a déjà vu que `len` renvoie la longueur de l'argument (qu'il s'agisse d'une chaîne de caractères, d'une liste...)

Essayer aussi `int('f2ae8',16), int('100110100010',2)`

La bibliothèque

On va rencontrer en Python une écriture ressemblant à celle du C ou de java pour l'accès aux champs des structures et unions : le point.

Ainsi, lorsqu'on écrit `maison.chaise`, on accède au champ "chaise" de la variable qui s'appelle "maison".

Ces champs peuvent être des fonctions (méthodes). On l'a déjà vu plus haut, quand on a fait `ma_liste.sort()` par exemple. On a appelé la *méthode* "sort" sur l'objet `ma_liste`.

D'autres exemples :

```
une_chaine="eviv bulgroz"
en_majuscule=une_chaine.upper() # met tout en majuscules
print en_majuscule
couleurs_html={"rouge":"#ff0000", "vert":"#00ff00", "bleu":"#0000ff", "blanc":"#ffffff",
"noir":"#000000"}
print couleurs_html.keys() # affiche la liste des clés (avant les :)
print couleurs_html.values() # affiche la liste des valeurs (après les :)
print couleurs_html.items() # affiche une liste avec des couples (clé,valeur)
for cle,valeur in couleurs_html.items(): print "en html, pour avoir du "+cle+", il faut
utiliser "+valeur
castors="riri&fifi&loulou"
liste_castors=castors.split("&") # crée une liste à partir d'une chaîne, en séparant avec
le(s) caractère(s) indiqué(s)
print liste_castors
print ", ".join(liste_castors) # l'inverse : concatène les éléments d'une liste en les
recollant avec la chaîne indiquée
```

Enfin, il y a un type d'objet particulier : les modules. Il s'agit de l'équivalent des bibliothèques du C.

Pour utiliser un module, on fait `import nom_du_module` et cela permet ensuite d'appeler des fonctions dans ce module, par exemple `nom_du_module.nom_de_la_fonction(parametres...)`

La documentation de tous les modules de base est disponible sur http://doc.python.org/lib/

Le module `sys`

```
import sys
print sys.argv # il s'agit des arguments du programme !
print "J'ai %d arguments, et le premier est %s"%(len(sys.argv),sys.argv[1])
sys.stderr.write("Ceci s'affiche sur l'erreur standard. Tapez votre nom, puis ENTREE, puis
Ctrl+D: ")
nom=sys.stdin.read()
sys.stdout.write("Bonjour, %s!\n"%nom)
# NB: sys.stdout.write et print, c'est presque pareil (au \n près!)
```

Manipulation de fichiers

```
f=open("toto") # ouverture en lecture
donnees=f.read(10) # lit 10 octets
encore_des_donnees=f.read() # lit autant d'octets que possible (tout le fichier)
f.close() # referme

ff=open("titi","w+") # le 2è argument est le mode, exactement comme pour la fonction C fopen
ff.write("bonjour, monde!\n")
ff.close()
```

Pour finir...

On veut écrire un programme `dico.py` qui va prendre comme arguments :
- un nom de fichier contenant un dictionnaire (le dictionnaire sera simplement une liste de mots, avec un mot par ligne ; comme ceux qu'on trouve dans `/usr/share/dict`)
- un ou plusieurs mots

Pour chaque mot listé sur la ligne de commande, il faudra afficher s'il appartient au dictionnaire, *ou pas*.

Exemple :

```
[~$] ./dico.py /usr/share/dict/french luke je suis ton père
luke n'est pas dans le dictionnaire.
je est dans le dictionnaire.
suis est dans le dictionnaire.
ton est dans le dictionnaire.
père est dans le dictionnaire.
[~$]
```

Si besoin est, il y a un dictionnaire ici. Ce dictionnaire est codé en ISO-8859, alors que votre système est probablement en UTF-8. Tester avec des mots accentués comme `déjà`. Que faut-il faire ?