DAUPHINE
UNIVERSITÉ PARIS

MASTER MATHÉMATIQUE, INFORMATIQUE, DÉCISION, ORGANISATION (MIDO)
2ÈME ANNÉE - SPÉCIALITÉS MIAGE-ID, MIAGE-IF, MIAGE-SITN

TP DE LANGAGE PYTHON 3

2014-2015

Maude Manouvrier

La rédaction de ce TP a été réalisé à l'aide des tutoriels et livres en ligne, cités à la fin de ce document, ainsi que du TP de M. Menceur qui faisait ce cours en 2010-2011.

Ce TP va vous permettre d'apprendre le langage **Python 3 par l'exemple**, à l'aide de petits exercices.

Contents

Python, développé depuis 1989 par Guido van Rossum et de nombreux contributeurs bénévoles, est un langage à typage dynamique (i.e. le type des objets manipulés n'est pas forcément connu à l'avance mais est défini ÃÂ partir de la valeur de la variable) et fortement typé (i.e. qu'il garantit que les types de données employés décrivent correctement les données manipulées). Il est doté 'd'une gestion automatique de la mémoire par ramasse-miettes (pas de gestion de pointeurs!!!) et d'un système de gestion d'exceptions.

En Python : tout est objet.

Le langage Python peut être interprété (interprétation du bytecode compilé) ou traduit en *bytecode*, qui est ensuite interprété par une machine virtuelle Python. Il est interfaçable avec des langages comme le C, le C++ ou Java.

Pour la réalisation de ce TP, vous êtes invités à lire ce qui suit dans le document, tout en vous aidant de la documentation en ligne et des ouvrages référencés à la fin du document et dont les liens sont accessibles à partir de la page Web : `http://www.lamsade.dauphine.fr/~manouvri/PYTHON/CoursPython_MM.html`.

1 Prise en main de Python

Cette section vous explique comment exécuter des commandes ou un programme Python soit en ligne de commande soit depuis un éditeur Python. Il est conseillé de tester les deux environnements.

Il est possible de programmer en Python en ligne de commande, c'est-à-dire en saisissant et en exécutant les instructions les unes à la suite des autres. Ceci se fait via un *interpréteur de commandes* (voir section 1.1). Il est également possible de saisir toutes les instructions d'un programme dans un fichier et d'exécuter ce programme (voir section 2).

Pendant les TP, vous aurez deux manières de programmer en Python: soit en utilisant l'interpréteur de commandes (en exécutant la commande `Python` depuis un terminal) et en saisissant vos programmes dans un éditeur de texte de votre choix, soit via l'outil IEP (*Interactive Editor for Python* - voir section 2) qui contient, dans la même interface, un interpréteur de commandes et un éditeur de texte. La section 2.1 vous indique comment installer Python sur votre machine personnelle et vous présente également l'interface graphique d'un outil (IDLE), l'environnement de développement intégré pour le langage Python.

1.1 Programmation Python en ligne de commande

Dans un interpréteur de commandes, le symbole >>> correspond au signal d'invite, ou *prompt* principal, lequel vous indique que Python est prêt à exécuter une commande. Les lignes non précédées de ce symbole correspondent à l'affichage d'un résultat. Après avoir saisi chaque instruction, il suffit de taper sur la touche `Enter` pour que la commande soit exécutée (i.e. interprétée).

Pour exécuter un programme tapé dans un fichier (d'extension `.py`), il suffit de saisir la commande suivante dans un terminal : `python MonProgramme.py`
Pour quitter l'interpréteur de commandes, il faut taper l'instruction `exit()` ou `quit()`.

2 L'outil IEP (*Interactive Editor for Python*)

L'interface de l'outil IEP[1] (*Interactive Editor for Python*) est composée de 3 parties (voir figure 1) : l'interpréteur de commandes en ligne (en haut de l'interface), un éditeur de texte pour écrire des programmes dans des fichiers (en bas à gauche, un onglet par programme) et un gestionnaire de fichiers (pour aller ouvrir un programme par

[1] http://www.iep-project.org/index.html

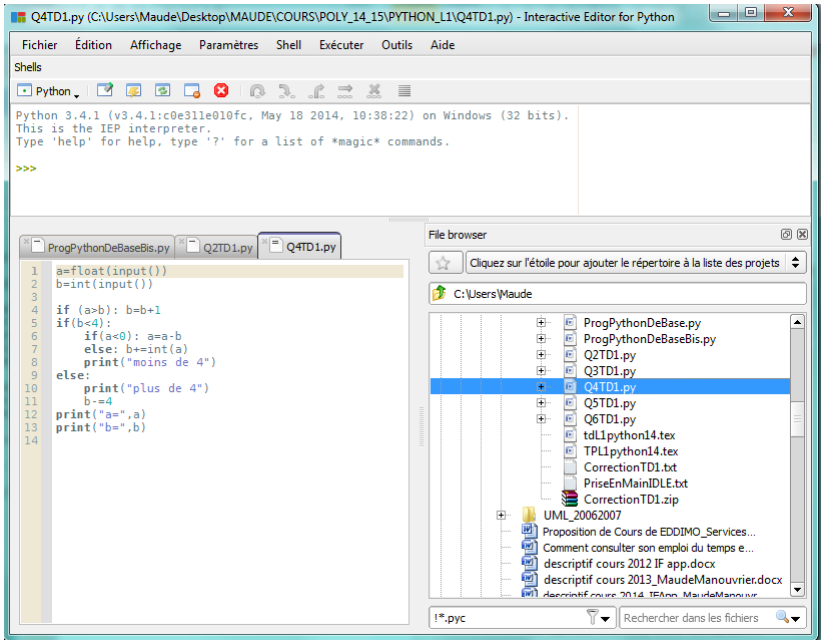

Figure 1: Interface de l'outils IEP.

exemple - en bas à droite).

Pour exécuter un programme, il suffit de cliquer sur l'onglet correspondant après l'avoir ouvert puis de sélectionner dans le menu Exécuter le fichier (ou Ctrl-E).

Pour installer ce logiciel sur votre machine personnelle, vous devez préalablement avoir installé Python (voir section suivante). Puis vous devez télécharger la version correspondant à votre machine à l'adresse (un fichier d'extension `.exe` sous windows) : `www.iep-project.org/downloads.html`. Attention, sous windows, le logiciel installe une icône sur le bureau.

2.1 Installer Python sur votre machine personnelle

Pour installer Python sur votre machine personnelle, vous devez télécharger la dernière version du langage à l'adresse `https://www.Python.org/downloads/`.

Sous windows par exemple, les instructions à suivre sont :

- Telecharger le fichier (d'extension .msi pour Windows installer) correspondant à la configuration de votre machine (voir figure 2 par exemple).
- Cliquer sur le fichier pour l'installer (voir figure 3).

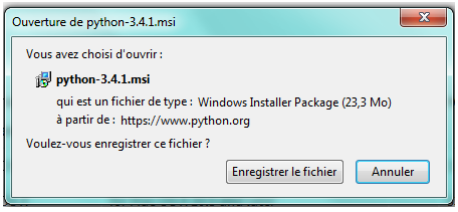

Figure 2: Téléchargement de l'interface IDLE sous Windows.

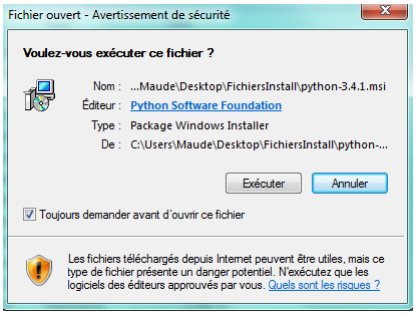

Figure 3: Installer l'interface IDLE sous Windows.

3 L'interface *IDLE (Python GUI)* fournie avec Python

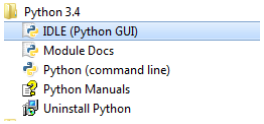

Figure 4: Lancer l'interface IDLE sous Windows.

L'installation de Python (voir section précédente) génère également l'installation d'une interface, appelée *IDLE (Python GUI)*. Cette interface vous permet de saisir des instructions en ligne de commande mais également d'exécuter des programmes Python enregistrés dans des fichiers. Contrairement à l'interface de l'outil *IEP* (voir section 2), l'interpréteur de commandes et l'éditeur de texte sont dans des fenêtres séparées.

Une fois la dernière version de Python installée, l'interface IDLE est disponible depuis le menu démarrer (repertoire Python x.y avec x.y le numéro de la version de Python installée). Il suffir de cliquer sur *IDLE (Python GUI)* – voir figure 4 – qui va vous ouvrir l'interface graphique (interpréteur de commandes en ligne) oÃ¹$vous pourrez tapervos instructions Python en ligne de commande$.

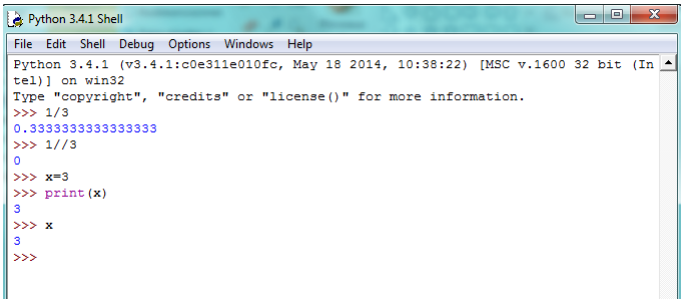

Figure 5: Programmer en ligne de commande via l'interface IDLE.

Pour écrire un programme dans un fichier, dans le menu *File*, sélectionnez *New File*. Une nouvelle fenêtre s'ouvre. Tapez votre programme Python dans cette fenêtre (attention aux indentations). Pour exécuter votre programme, allez dans le menu Run et faites Run Modules (ou F5) – voir figure 7. Il va vous être demandé de faire une sauvegarde de votre fichier (qui a généralement l'extension `.py`) – voir figure 6 –, puis votre programme s'exécutera (dans la fenêtre en ligne de commande précédemment ouverte). Le début de l'execution de votre programme est indiqué dans la fenêtre en ligne de commande par:
`========== RESTART ==========`

Dans le menu File vous pouvez également ouvrir un programme existant. Pour passer d'une fenêtre à une autre, vous pouvez utiliser le menu Windows – voir figure 8.

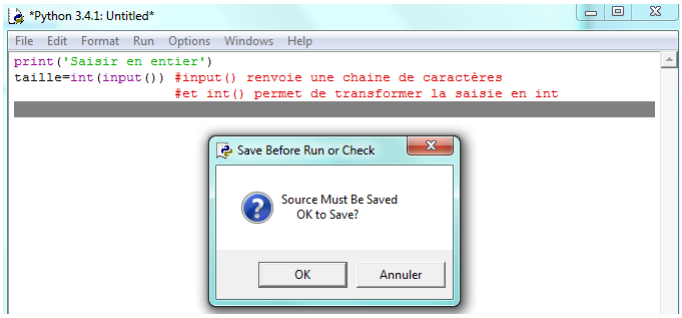

Figure 6: Enregistrer un programme Python dans un fichier (d'extension `.py`) via l'interface IDLE.

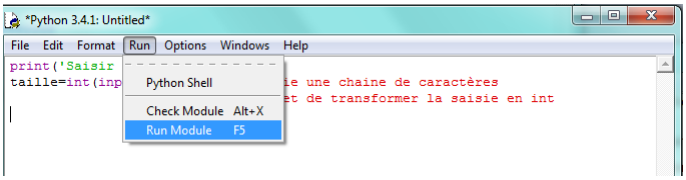

Figure 7: Exécuter un programme Python depuis un fichier (d'extension `.py`) via l'interface IDLE.

4 Premiers pas en Python

Cette section présente quelques exemples de code Python, réalisés avec `Python 3.4` en ligne de commande.

Les lignes commençant par >>> correspondent aux instructions. Les lignes situées juste en dessous correspondent à l'affichage après exécution de l'instruction (i.e. après avoir tapé `<Enter>`).

En Python, les commentaires commencent par le symbole `#`.

Vous êtes invités à taper les exemples ci-dessous pour vous entaîner et à répondre à chaque question associée aux exemples.

4.1 Faire des calculs avec Python

Exercice 1 *:* **Quelques exemples de calcul**
Essayez, en les exécutant, de comprendre ce que fait chaque instruction (non commentée) de l'exemple ci-dessous. Cet exemple est valable en ligne de commande uniquement.
En ligne de commande, la valeur d'une variable ou le résultat d'un calcul s'affiche directement après la saisie de cette variable ou de ce calcul.

```
>>> 5+3
8
>>>5*3
15
>>>5**3
125
>>> x=1 # declaration d'un variable x de valeur 1 (# pour le commentaire)
>>> x   # affichage de x
1
>>> a,b,c=3,5,7 # declaration de 3 variables a, b et c de valeurs resp. 3, 5 et 7
>>> a-b/c
2.2857142857142856
>>> (a-b)/c
-0.2857142857142857
```

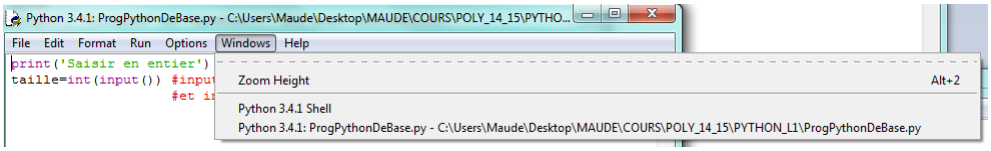

Figure 8: Gestion des fenêtres sous l'interface IDLE.

```
>>> b/c
0.7142857142857143
>>> b//c
0
>>> b%c
5
>>> d=1.1
>>> d/c
0.15714285714285717
>>> d//c
0.0
```

Exemple 1 *:* **Importation de la librairie mathématique et exemple de fonction mathématique**

```
>>> from math import *  # Pour importer la librairie de fonctions mathematiques
>>> sqrt(4) # Pour calculer la racine carree
2.0
>>>pi
3.141592653589793
```

NB: La liste des fonctions de la librairie `math` est disponible à l'adresse :
`http://docs.python.org/library/math.html?highlight=math#math`

4.2 Affichage

Exemple 2 *:* **Utilisation de la fonction d'affichage `print()`**

```
>>> print(a+b) # a et b sont les variable de l'exercice 1
8
>>> print('la valeur de', a,'+',b,'est :', a+b)
('la valeur de', 3, '+', 5, 'est :', 8)
```

4.3 Déclaration et initialisation de variables et types

Exemple 3

```
>>> print(type(a)) # a est la variable de l'exercice 1
<class 'int'>
>>> pi=3,14
>>> print(type(pi))
<class 'tuple'>
>>> pi=3.14
>>> print(type(pi))
<class 'float'>
>>> s='exemple de chaine de caracteres'
>>> type(s)
<class 'str'>
```

```
>>> 2+'1.5'
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> 2+eval('1.5') # Pour \'eliminer l'erreur pr\'ec\'edente
3.5
```

4.4 Chaînes de caractères

Exemple 4 *:* **Manipulation des chaîne de caractères et exemples de fonctions sur les chaînes de caractères**

```
>>> s='un exemple de chaine'
>>> s2="un autre exemple"
>>> s[1] # Acces au caractere d'indice 1 (les indices commencent a zero)
'n'
>>> print(s[0],s2[0])
u u
>>> print(s[4],s2[0])
x u
>>> print(s + ' et ' + s2) # Concatenation de chaines
un exemple de chaine et un autre exemple
>>> s3=s + ' et ' + s2
>>> s3
'un exemple de chaine et un autre exemple'
>>> s2*2
'un autre exempleun autre exemple'
>>> print('La taille de s est :', len(s))
La taille de s est : 20
>>> s3[0:3] # Recuperation des caracteres de position entre les 0 et 3e
'un '
>>> s3[4:8]
'xemp'
>>> print(s3[:3]) # Recuperation des 3 premiers caracteres
un
>>> print(s3[3:]) # Recuperation des caracteres a partir de la position 3
exemple de chaine et un autre exemple
>>> s3[::-1]
'elpmexe ertua nu te eniahc ed elpmexe nu'
>>> s3.find("exemple")
3
>>> s3.replace("chaine","str")
'un exemple de str et un autre exemple'
>>> help(str) # pour afficher l'aide sur la classe str
```

Exemple 5 *:* **Exemple de récupération des mots d'une chaÃ®ne de caractères**

```
>>> sentence = 'It is raining cats and dogs'
>>> words = sentence.split()
>>> print(words)
['It', 'is', 'raining', 'cats', 'and', 'dogs']
```

4.5 Boucles et conditions

A partir d'ici, vous pouvez commencer à saisir les instructions dans un fichier d'extension `.py` et vous pouvez exécuter ce fichier.

Attention : En Python il n'y a pas, comme dans certains langages, d'accolade ouvrante ou fermante pour délimiter un bloc d'instructions. Les blocs d'instructions en python sont délimités par `":"` puis des tabulations : toutes les instructions consécutives à un `":"` et débutant par un même nombre de tabulations appartiennent à un même bloc d'instructions.

Exercice 2 *:* **Boucle for**
Tapez le code suivant et observez le résultat.

```
for i in range(10): # Ne pas oublier les deux points!!
     x = 2 # Attention ne pas oublier une tab. en debut de ligne sinon erreur!!!
     print(x*i) # Ne pas oublier la tabulation en debut de ligne!!
# Tapez encore une fois <Enter> si vous Ãªntes en ligne de commande
```

Exercice 3 *:* **Boucle while**
Tapez le code suivant et observez le résultat.

```
a=0
while(a<12):  # Ne pas oublier les deux points!!
   a=a+1 # Ne pas oublier la tabulation en debut de ligne!!
   print(a, a**2,a**3) # Ne pas oublier la tabulation en debut de ligne!!
# Tapez encore une fois <Enter> si vous Ãªtes en ligne de commande
```

Exercice 4 *:* **Condition If/Then/Else**
Tapez le code suivant et observez le résultat.

```
a=0
if a==0:  # Ne pas oublier les deux points!!
  print('0')  # Ne pas oublier la tabulation en debut de ligne!!
elif a==1: # Ne pas mettre de tabulation et ne pas oublier les deux points!!
  print('1') # Ne pas oublier la tabulation en debut de ligne!!
else:  # Ne pas mettre de tabulation et ne pas oublier les deux points!!
  print('2') # Ne pas oublier la tabulation en debut de ligne!!
# Tapez encore une fois <Enter> si vous Ãªtes en ligne de commande
```

4.6 Récupérer des saisies claviers

Exemple 6

```
>>> s=input() # taper <Enter> puis saisir quelque chose au clavier
>>> s
>>> s=input("Saisir une chaine :") # taper <Enter>
Saisir une chaine :
>>> s
```

Exercice 5 *: Ecrivez un script permettant d'obtenir le résultat suivant*

```
Saisissez une chaine
une chaine
La chaine inversee est:
eniahc enu
```

Exercice 6 *: Récupérez, analysez et exécutez le fichier*
http://www.lamsade.dauphine.fr/~manouvri/PYTHON/EXEMPLES/SaisieEntier.py

5 Structures de données

5.1 Listes

Exercice 7 *:* **Manipulation des listes**
Tapez chacune des insctructions suivantes (en ligne de commande) et observez le résultat.

```
>>> list=['lundi', 2, 'janvier']
>>> print(list)
>>> list[0] # Mettre print(list[0]) si vous n'Ãªtes pas en ligne de commandes
>>> list[-1]
>>> print(list[2])
>>> len(list)
>>> list.append(2010)
>>> list
>>> list[3]=list[3]+1
>>> list[3]
>>> del list[0]
>>> list
>>> list.insert(0,'mardi')
>>> list
>>> 'mardi' in list
>>> 'lundi' in list
>>> list.index("mardi")
>>> list2=list[1:3]
>>> list2
>>> list3=list[:2]
>>> list3
>>> list4=list[1:]
>>> list4
>>> list5=list[-3:-1]
>>> list5
>>> list6 = list[:-1]
>>> list6
>>> list3=list3 + [2011]
>>> list3
>>> list7=3*list
>>> list7
>>> list.extend([3,4])
>>> list
>>> list=list.pop(0)
>>> list
>>> list=[1,2,3]
>>> list2=list # Attention list et list2 correspondent a la meme liste!!
>>> list
>>> list.pop(1)
>>> list
>>> list2
>>> list=[1,2,3]
>>> list2=list.copy() # list2 est une copie de list
>>> list.pop(1)
>>> list
>>> list2
```

```
>>> list=[1,"ab",[1,True]] # liste imbriquee
>>> list[2]
>>> print(list(range(10)))
>>> print(list(range(1,10)))
>>> print(list(range(1,10,3)))
>>> help(list) # pour l'aide sur les listes
```

Exemple 7 *:* **Saisies d'une liste au clavier**

```
>>> list = [input(),input(),input()]
>>> list
>>> list=[x for x in input("Saisir les elements de la liste separes
par une virgule (ex. 1,2,"abc") :").split(',')] # Taper <Enter>
>>> list
>>> list=[int(x) for x in input("Saisir des entiers separes
par une virgule (ex. 1,2,"abc") :").split(',')] # Taper <Enter>
>>> list
```

Exercice 8 *: Ecrivez une programme qui demande à l'utilisateur d'entrer des notes d'élèves. Si l'utilisateur entre une valeur négative, le programme s'arrête. En revanche, pour chaque note saisie, le programme construit progressivement une liste. Après chaque entrée d'une nouvelle note (et donc à chaque itération de la boucle), il affiche le nombre de notes entrées, la note la plus élevée, la note la plus basse, la moyenne de toutes les notes.*
Astuce : Pour créer une liste vide, il suffit de taper la commande `list = []`*.*

5.2 Dictionnaire

Exercice 9 *: Tapez le code suivant*[2] *et observez le résultat.*

```
>>> dico = {} # dictionnaire vide
>>> dico['computer'] = 'ordinateur'
>>> dico['mouse'] ='souris'
>>> dico['keyboard'] ='clavier'
>>> print(dico)
>>> print(dico.keys())
>>> print(dico.values())
>>> del dico['mouse']
>>> print(dico)
>>> print(len(dico))
>>> dico.__contains__('computer')
>>> print(dico.items())
>>> for clef in dico: # Ne pas oublier les 2 points
...     print(clef) # Attention ne pas oublier la tabulation!!
>>> for clef in dico: # Ne pas oublier les 2 points
...     print (clef, dico[clef]) # Attention ne pas oublier la tabulation!!
>>> for clef, value in dico.items(): # Ne pas oublier les 2 points
...     print(clef, value) # Attention ne pas oublier la tabulation!!
>>> dico2={'ordinateur': 'computer', 'souris' : 'mouse'}
>>> print(dico2)
>>> dico2=dico # Attention dico et dico2 correspondent au mÃªme dictionnaire!
>>> print(dico)
>>> del dico2['computer']
>>> print(dico)
```

[2] Repris et adapté de *Apprendre à programmer avec Python* de Gérard Swinnen, 2009

```
>>> print(dico2)
>>> dico3=dico.copy() # dico3 est une copie du dictionnaire reference par dico!
>>> del dico3['keyboard']
>>> print(dico)
>>> print(dico3)
>>> help(dict) # pour l'aide sur les dictionnaires
```

6 Références et adresses

En python, tout est objet, y compris les valeurs. Les variables sont des références à des valeurs.

Exemple 8 *:* **Références et adresses**

```
>>> help(id) # pour afficher la description de la fonction retournant une adresse
>>> id(2) # on affiche l'adresse de la valeur 2
10455072
>>> a=2
>>> id(a)
10455072 # la variable 'a' a la mÃªme adresse que celle de la valeur 2
>>> b=2
>>> id(b)
10455072 # la variable 'b' a la mÃªme adresse que celle de la valeur 2
>>> c=b
>>> id(c)
10455072 # la variable 'c' a la mÃªme adresse que celle de la valeur 2
>>> d=b
>>> id(d)
10455072 # la variable 'd' a la mÃªme adresse que celle de 'b'
>>> d=3
>>> id(d)# l'adesse de 'd' a changee
10455104
>>> id(3)
10455104
>>> list=[1,2,3]
>>> id(list)
140482820775816
>>> list2=list
>>> id(list2)
140482820775816 # list et list2 ont la mÃªme adresse
>>> id([1,2,3])
140482820774664 # Adresse de la liste [1,2,3] differente de celle de list et list2
>>> list3=[1,2,3]
>>> id(list3)
140482820772552 # adresse differente de celle de list, list2 et [1,2,3]
>>> id(list[0])
10455040
>>> id(list3[0])
10455040 # adresse de list[0] et list3[0]
>>> id(1)
10455040 # adresse de la valeur 1
```

```
>>> list is list2 # Pour tester l'identite memoire
True
>>> list[0] is 1
True
>>> list is list3
False # list et list3 correspondent a la mÃªme liste
      # mais a 2 emplacement memoire differents
>>> list[0] is list3[0]
True
```

Un exemple de représentation en mémoire d'une liste est donné dans la figure[3] 9.

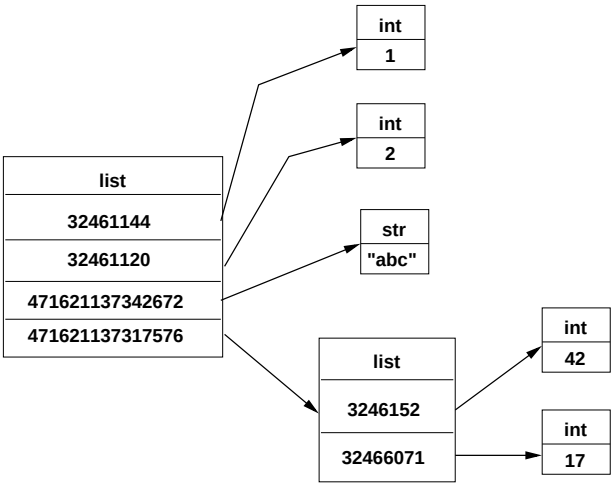

Figure 9: Représentation mémoire de la liste imbriquée `[1,2,"abc",[42,17]]`.

7 Fonctions

7.1 Fonctions Python existantes

La liste des fonctions Python existantes est disponible en ligne à l'adresse :
`https://docs.python.org/3/library/functions.html`

7.2 Fonction simple sans paramètre

Exercice 10 *:* **Définir une fonction sans paramètre**
Tapez le code suivant et observez le résultat.

```
>>> def function(): # Definition de fonction sans parametre - Ne pas oublier les :
...     n=10 # Faire une tabulation en debut de ligne
...     while n>0: # Faire une tab. en debut de ligne et ne pas oublier les :
...             print(n/2, n%2) # Faire 2 tabulations en debut de ligne
...             n=n-1  # Faire 2 tabulations en debut de ligne
# Tapez encore une fois <Enter> si vous etes en ligne de commande
>>>function() # Appel de la fonction
```

[3] Reprise et adapté de `http://judicael.courant.free.fr/pro/2014/ic1a/cours/05-memoire.poly.pdf`

Exercice 11 *:* **Définir une fonction sans paramètre qui appelle une autre fonction**
Tapez le code suivant et observez le résultat.

```
>>> def function2(): # Ne pas oublier les : et une tab. sur la ligne suivante
...     print('Affichage du resultat et du reste de la division des entiers de 10 \
...     function() # Faire une tabulation en debut de ligne
# Tapez encore une fois <Enter> si vous etes en ligne de commande
>>> function2()
```

7.3 Fonction avec paramètres

Exercice 12 *:* **Définir une fonction à 1 paramètre**
Tapez le code suivant et observez le résultat.

```
def function(n): # Definition d'une fonction a un parametre
...     while n>0: # Faire une tab. en debut de ligne et ne pas oublier les :
...             print(n/2, n%2) # Faire 2 tabulations en debut de ligne
...             n=n-1 # Faire 2 tabulations en debut de ligne
# Tapez encore une fois <Enter> si vous etes en ligne de commande
>>> function(3) # Appel de la fonction avec le parametre 3
# Tapez encore une fois <Enter> si vous etes en ligne de commande
>>> function(5) # Appel de la fonction avec le parametre 5
```

Exercice 13 *:* **Définir une fonction à 2 paramètres**
Tapez le code suivant et observez le résultat.

```
>>> def function(entree,diviseur): # Definition d'une fct. a plusieurs parametres
...     while(entree>0):
...             print(entree/diviseur, entree%diviseur)
...             entree=entree-1
# Tapez encore une fois <Enter> si vous etes en ligne de commande
>>> function(10,2)
```

Exercice 14 *: Tapez le code suivant et observez le résultat.*

```
>>> def afficher3fois(arg):
...     print(arg, arg, arg)
# Tapez encore une fois <Enter> si vous Ãªtes en ligne de commande
>>> afficher3fois(3)
>>> afficher3fois('exemple')
>>> afficher3fois([3,4])
>>> afficher3fois(3*4)
```

Exercice 15 *: Récupérez le fichier*
http://www.lamsade.dauphine.fr/~manouvri/PYTHON/EXEMPLES/CalculVolumeSphere.py
étudiez le code en ouvrant le fichier et exécutez le.

7.4 Valeur par défaut des paramètres

Exercice 16 *: Tapez le code suivant et observez le résultat.*

```
>>> def function(entree,diviseur=2): # fct. avec valeur par defaut pour diviseur
...     while(entree>0):
...             print(entree/diviseur, entree%diviseur)
...             entree=entree-1
# Tapez encore une fois <Enter> si vous Ãªtes en ligne de commande
>>> function(10)
```

Exercice 17 *: Tapez le code suivant et observez le résultat.*

```
>>> def function(entree=10,diviseur=2):
...     while(entree>0):
...             print(entree/diviseur, entree%diviseur)
...             entree=entree-1
# Tapez encore une fois <Enter> si vous Ãªtes en ligne de commande
>>> function()
>>> function(4,2)
>>> function(diviseur=4,entree=12)
```

7.5 Affecter une instance de fonction à une variable

Exercice 18 *: Tapez le code suivant et observez le résultat.*

```
>>>def multiplication(n,p):
...     return n*p # pour retourner une valeur
# Tapez encore une fois <Enter> si vous Ãªtes en ligne de commande
>>> a=multiplication(3,6)
>>> a
```

7.6 Fonction avec un nombre variable de paramètres

Exemple 9 *:*

```
# les valeurs des arguments saisis vont etre stockes dans un t-uple - cf. print(type
def fonction_test(*args):
    print("type de args :",type(args))
    for arg in args:
        print("paramètre:", arg)
        print()

fonction_test()
fonction_test(1)
fonction_test(1,"a")
fonction_test(1,"a",3)
```

Exercice 19 *: Ecrire une focntion qui calcule la somme des entiers passés en paramètres.*

7.7 Passage des paramètres : immuable et non immuable

En Python, il existe en effet deux types dâobjets: les non immuables ou *mutables* tels que les listes, les dictionnaires, etc., que l'on peut modifier après leur création, et les immuables ou *non mutables* tels que les strings, int, floats, tuples, etc., que l'on ne peut pas modifier aprés leur création.

Exercice 20 *: Tapez le code de cette mauvaise fonction* `echange` *dans un fichier, exécutez le et analysez le résultat.*

```
def incr(a):
    print("adresse a: ",id(a))
    a=a+1
    print("Dans la fonction a =",a)

def ajouter(l):
    l=l.append(5)
```

```
a=3
print("adresse a: ",id(a))
incr(a)
print("Apres l'appel a=",a)

l=[1,2,3]
ajouter(l)
print("l=: ",l)
```

7.8 Variable locale/variable globale

Exemple 10 *:* **Mauvaise fonction `echange`**

```
def echange(a,b):
    print("adresses des parametres : ", id(a),id(b))
    c=a
    a=b
    b=c

x,y = 2,3
print("adresses de x et de y  : ", id(x),id(y))
echange(x,y)
print("x=",x)
print("y=",y)
```

Si vous exécutez le code précédent, vous verrez que les variables `x` et `y` n'ont pas été échangées car il s'agit de variables non immuables.

Exemple 11 *:* **Exemple de fonction `echange` qui fonctionne**

```
def echange2(a,b):
    c=a
    a=b
    b=c
    return a,b

x,y = 2,3
x,y=echange2(x,y) # les valeurs de x et y sont bien echangees car
                  # on les modifie leur valeur ici
print("x=",x)
print("y=",y)
print() # pour sauter une ligne

a=2
b=3
a,b=echange2(a,b) # les valeurs de a et b sont bien echangees car
                  # on les modifie leur valeur ici
print("a=",a)
print("b=",b)
print()
```

Exemple 12 *:* **Autre exemple de fonction `echange` qui fonctionne**

```
def echangeab():
    global a # a est la variable globale du programme
    global b # b est la variable globale du programme
    c=a
    a=b
    b=c

a=2
b=3
echangeab()
print("a=",a)
print("b=",b)
```

7.9 Fonction anonyme (lambda function)

Python permet la création de fonctions anonymes (i.e. sans nom et donc non définie par `def`) à l'aide du mot-clé `lambda`. Une fonction anonyme ne peut pas avoir d'instruction `return` et doit forcément retourner une expression. De telles fonctions permettent de représenter tout sous forme de fonctions sans réllement en définir explicitement.

Exercice 21 *: Tapez le code*[4] *suivant et observez le résultat.*

```
>>> def f (x): return x**2
>>> print(f(8))
>>> g = lambda x: x**2
>>> print(g(8))
>>> (lambda x: x*2)(3) //2
>>> def make_incrementor (n): return lambda x: x + n
>>> f = make_incrementor(2)
>>> g = make_incrementor(6)
>>> print(f(42), g(42))
>>> print(make_incrementor(22)(33))
```

8 Fichiers

8.1 Instanciation du répertoire courant

Exemple 13 *: modifier le répertoire courant de l'interpréteur*

```
>>>from os import chdir
>>>chdir("/home/login/exercices") # Mettre ici le repertoire
                                  # de stockage de vos exercices
```

La première ligne permet d'importer la commande `chdir()` du module `os` contenant une série de fonctions permettant de dialoguer avec le système d'exploitation (os = operating system).
La deuxième ligne permet de spécifier le répertoire courant de l'interpréteur python, i.e. le répertoire où il va récupérer les fichiers.

[4] Repris et adapté de `http://www.secnetix.de/olli/Python/lambda_functions.hawk`

8.2 Manipulation de fichiers

Exercice 22 *: Tapez le code suivant et observez le résultat.*

```
>>> f=open('test.txt','w') # Pour ouvrir un fichier en mode ecriture
>>> f.write("Bonjour\n") # Pour ecrire dans le fichier
>>> f.write("Ceci est un test d'ecriture dans un fichier")
>>> f.close() # Pour fermer le fichier
              " verifier le fichier dans un editeur de texte
>>> f=open('test.txt','r') # # Pour ouvrir un fichier en mode lecture
>>> b=f.read() # Pour lire tout le fichier
>>> print(b)
>>> f.close() # Pour fermer le fichier
>>> f=open('test.txt','r') # # Pour ouvrir un fichier en mode lecture
>>> b=f.read(3) # Pour lire 3 caracteres
                # a partir de la position courante du curseur
>>> print(b)
>>> f.close()
>>> f=open('test.txt','r')
>>> b = f.readline() # Pour lire une ligne dans le fichier
>>> print(b)
>>> b = f.readlines() # Pour lire toutes les lignes et les stocker dans une liste
>>> print(b)
>>> f.close()
>>> f=open('test.txt','a') # Pour ouvrir le fichier en mode ajout
>>> f.write("\n Fin")
>>> f.close()
>>> f=open('test.txt','r')
>>> b = f.readlines()
>>> print(len(b))
>>> print(b)
>>> f.close()
>>> f=open('test.txt','r')
>>> print(f.read()) # Pour lire tout le fichier
                   # deplace le curseur Ã  la fin du fichier
>>> print(f.read())
>>> f.close()
```

8.3 Copie de fichiers

Exercice 23 *: Récupérez le fichier*
http://www.lamsade.dauphine.fr/~manouvri/PYTHON/EXEMPLES/CopieFichier.py
Etudiez le code en ouvrant le fichier dans un éditeur de texte et exécutez le.

8.4 Copier des variables dans un fichier

Exercice 24 *: Tapez le code suivant*[5] *et observez le résultat.*

```
>>> a = 5
>>> b = 2.83
>>> c = 67
>>> f = open('test.txt', 'w')
```

[5] Repris de *Apprendre à programmer avec Python* de Gérard Swinnen, 2009

```
>>> f.write(str(a)) # Pour ecrire un entier converti en string
>>> f.write(str(b))
>>> f.write(str(c))
>>> f.close()
>>> f = open('test.txt', 'r')
>>> print(f.read())
>>> f.close()

>>> import pickle # Pour importer un module permettant
                  # de conserver le type des variables
>>> f = open('Monfichier', 'wb') # ouverture d'un fichier binaire
>>> pickle.dump(a, f) # Pour enregistrer dans le fichier
>>> pickle.dump(b, f) # equivalent a write mais en conservant le type de b
>>> pickle.dump(c, f)
>>> f.close()
>>> f = open('Monfichier', 'rb')
>>> print(f.read())
>>> f.close()
>>> f = open('Monfichier', 'rb')
>>> t = pickle.load(f) # Operation inverse de dump
>>> print(t, type(t))
>>> t = pickle.load(f) # Equivalent a read mais en conservant le type de b
>>> print(t, type(t))
>>> t = pickle.load(f)
>>> print(t, type(t))
>>> f.close()
```

9 Gestion des exceptions

Exercice 25 *: Récupérez le fichier*
http://www.lamsade.dauphine.fr/~manouvri/PYTHON/EXEMPLES/ExceptionOuvertureFichier.py
étudiez le code en ouvrant le fichier dans un éditeur de texte et exécutez le.

10 Programmation orientée-objet

10.1 Premier exemple de classe

Exercice 26 *: Récupérez le fichier*
http://www.lamsade.dauphine.fr/~manouvri/PYTHON/EXEMPLES/ExempleClasse1.py
étudiez le code en ouvrant le fichier et exécutez le.

Exercice 27 *: Tapez le code suivant et observez le résultat.*

```
>>> import ExempleClasse1 *
>>> c1 = ExempleClasse1.CompteBancaire('Toto', 1000)
>>> c1.depot(350)
>>> c1.retrait(200)
>>> c1.affiche()
>>> print(c1.nom)
>>> print(c1)
```

10.2 Accessibilité

Exercice 28 *: Reprendre la classe de l'exercice précédent et remplacer* `nom` *par* `__nom`*. Exécutez à nouveau l'exemple de l'exercice précédent. Observez le résultat.*

10.3 Objet complexe

Exercice 29 *: Récupérez le fichier*
`http://www.lamsade.dauphine.fr/~manouvri/PYTHON/EXEMPLES/rect_carre.py`
étudiez le code en ouvrant le fichier et exécutez le.

10.4 Héritage

Exercice 30 *: Récupérez le fichier*
`http://www.lamsade.dauphine.fr/~manouvri/PYTHON/EXEMPLES/formes.py`
étudiez le code en ouvrant le fichier et exécutez le.

NB: L'héritage multiple est possible en Python.

11 Documentation en ligne et liens importants

- **Tutoriel de la documentation officielle en ligne** : `http://docs.python.org/tutorial/index.html`
- **Livre et exercices corrigés en ligne sur Python 2 et 3** : *Apprendre à programmer avec Python* de Gérard Swinnen, 2009 - `http://inforef.be/swi/python.htm`
- **Livre (format pdf) en ligne sur Python 3:** *Introduction à Python 3* de Robert CORDEAU, 2010 `http://www.afpy.org/Members/bcordeau/Python3v1-1.pdf/download`
- **Tutoriel en ligne pour développer en Python sous Eclipse**: *Python Development with PyDev and Eclipse - Tutorial* de Lars Vogel, 2011 - `http://www.vogella.de/articles/Python/article.html`
- **Tutoriel en ligne** : *A Quick, Painless Tutorial on the Python Language* de Norman Matloff
  - Tutoriel en ligne (2008) : `http://heather.cs.ucdavis.edu/~matloff/Python/PythonIntro.html`
  - Fichier pdf (2009) : `http://mononeurona.org/files/userfiles/aarkerio_121.pdf`
- **Tutoriel en ligne** : `http://www.tutorialspoint.com/python/index.htm`
- **Documentation en ligne** : `http://infohost.nmt.edu/tcc/help/lang/python/docs.html` dont *Extending and Embedding Python* de Guido van Rossum, Fred L. Drake, Jr., editor October 06, 2009, Python Software Foundation (`http://infohost.nmt.edu/tcc/help/lang/python/2_6_3/extending.pdf`)
- **Site officiel python** : `http://www.python.org/`
- **Association francophone Python** : `http://www.afpy.org/python/tutoriels`
- **Documentation en ligne de l'intégration C/C++/Python** : `http://docs.python.org/extending/extending.html`