I Les listes

1. Création d'une liste

Une liste est une séquence d'objets qui peuvent être de types différents (y compris des listes). Une liste s'écrit entre deux crochets et ses éléments sont séparés par une virgule. La liste vide s'écrit `[]`. Une liste peut être définie en écrivant tous les éléments où par une formule (méthode « en compréhension ») :

```
>>> liste1 = [1,2.5,[1,2],'psi']
>>> type(liste1) # donne le type d'un objet
<type 'list'>
>>> liste2 = [2*i+1 for i in range(10)] # impairs < 20
>>> print(liste2)   # parenthèses obligatoires en Python 3
[1,3,5,7,9,11,13,15,17,19]
>>> liste3 = range(0,20,2) # les pairs cette fois
>>> print(liste3)
range(0, 20, 2)
>>> liste3 = list(liste3) # transformation d'un objet range en liste
>>> print(liste3)
[0,2,4,6,8,10,12,14,16,18]
```

La fonction `range(début, fin, pas)` permet de générer des listes d'entiers de `début` (inclus) à `fin` (exclus) avec le `pas` demandé. Si `début` est omis, sa valeur est 0, si le `pas` est omis, sa valeur est 1.

2. Opérations sur une liste

Les éléments d'une liste sont indicés à partir de 0. L'élément d'indice i d'une liste s'écrit `liste[i]`, les listes sont des objets **mutables**, on peut donc modifier les éléments individuellement (contrairement aux n-uplets ou tuples comme nous le verrons plus tard) :

```
>>> liste=[1,2.5,[1,2],'psi']
>>> len(liste) # longueur de la liste
4
>>> print(liste)
[1,2.5,[1,2],'psi']
>>> liste[1] = 3
>>> print(liste)
[1,3,[1,2],'psi']
```

Pour extraire une sous-liste d'une liste on utilise le « slicing » (tranche) :

```
>>> liste = [1,2,[1,2],'psi',4.2]
>>> liste[:3]            # les 3 premiers
[1,2,[1,2]]
>>> print(liste[1::2])   # de 2 en 2 à  partir du deuxième
[2,'psi']
>>> print(liste[1:])     # tout à  partir du deuxième
[2,[1,2],'psi',4.2]
>>> print(liste[-3:])    # les 3 derniers
[[1,2],'psi',4.2]
>>> print(liste[3:0:-1]) # du 4ième au 2ième en sens inverse
['psi',[1,2],2]
```

Pour rajouter des éléments :

```
>>> liste1 = ['lundi','mardi','mercredi']
>>> liste2 = ['jeudi','vendredi']
>>> liste3 = liste1+liste2        # concaténation
>>> print(liste3)
```

```
['lundi','mardi','mercredi','jeudi','vendredi']
>>> liste4 = [0]*7                    # répétition
>>> print(liste4)
[0,0,0,0,0,0,0]
>>> print(liste4.append(1)) # rajoute 1 à la fin de la liste
[0,0,0,0,0,0,0,1]
```

Pour insérer un objet à l'indice i on utilisera le « slicing » :

```
>>> Liste=[0,1,2,3]
>>> Liste[2:2]=[5,6] # insertion de 5 et 6 à l'indice 2 (sans écraser d'éléments)
>>> print(Liste)
[0,1,5,6,2,3]
>>> Liste[1:4] = [8,8] # remplacement des éléments d'indices 1 à 3
>>> print(Liste)
[0,8,8,2,3]
```

Suppression d'un élément (ou une tranche) :

```
>>> Liste = [0,1,2,3,4,2,5]
>>> print(Liste.pop()) # renvoie et supprime le dernier élément
5
>>> print(Liste)
[0,1,2,3,4,2]
>>> 4 in Liste # vérification de l'appartenance
True
```

Copier une liste :

```
>>> L1 = [1,2,3]
>>> L2 = L1 # on copie la liste entière (semble-t-il)
>>> L1[0] = 4
>>> print(L2)
[4, 2, 3]
>>> L1 = [1,2,3]
>>> L2 = L1[:] # on copie les éléments un par un
>>> L1[0] = 4
>>> print(L2)
[1, 2, 3]
>>> L1 = [1,2,4]
>>> L2 = [k for k in L1]  # balayage des éléments de la liste
>>> L1[0] = 0
>>> print(L2)
[1, 2, 4]
```

II Les chaînes de caractères

Dans une première approche, une chaîne de caractères peut être vue comme une suite de caractères indicés en commençant par l'indice 0. En Python 3, les chaînes de caractères sont des chaînes « unicodes », ce qui permet notamment une gestion facile des caractères accentués.

1. Écriture d'une chaîne

Une chaîne de caractères en Python peut être délimitée par des apostrophes (simple quotes) ou bien par des guillemets (double quotes). Exemple :

```
>>> chaine1 = "C'est␣un␣premier␣message."
>>> chaine2 = 'Message␣"deux".'
>>> chaine3 = 'C\'est␣un␣autre␣message.'
>>> print(chaine1,chaine2,chaine3)
C'est␣un␣premier␣message.␣Message␣"deux".␣C'est un autre message.
```

```
>>> type(chaine1)
str
```

L'instruction `print` insère un espace entre chaque chaîne. On notera dans la troisième variable, comment insérer l'apostrophe dans une chaîne délimitée par deux apostrophes, grâce au caractère d'échappement \ (`backslash`). On peut le rencontrer dans plusieurs circonstances à l'intérieur d'une chaîne, notamment :

- `\'` : pour insérer l'apostrophe dans une chaîne délimitée par deux apostrophes,
- `\"` : pour insérer un guillemet dans une chaîne délimitée par deux guillemets,
- `\n` : pour insérer un saut de ligne,
- `\t` : pour insérer une tabulation.

Pour insérer le caractère backslash il suffit de le « doubler » :

```
>>> c1 = 'C\'est␣une␣chaine␣avec␣\\'
>>> print(c1)
C'est␣une␣chaine␣avec␣\
```

Notons enfin qu'il est possible de délimiter une chaîne par des triples apostrophes (ou guillemets), ce qui permet de l'écrire sur plusieurs lignes. Ceci est souvent utilisé pour documenter les fonctions lors de leur définition.

La plupart des systèmes d'exploitation modernes utilisent l'unicode pour l'encodage des caractères, en particulier la norme `utf-8`. L'unicode permet d'encoder non seulement les caractères accentués, mais également d'autres symboles et même d'autres alphabets. Il est toutefois possible de convertir une chaîne unicode en une chaîne d'octets (type `bytes`), on peut utiliser la méthode `encode` avec le paramètre `'utf8'` ; inversement, la méthode `decode('utf8')` permet de reconstruire une chaîne de caractères unicode.

```
>>> nom1='Hélène'
>>> print(nom1)
Hélène
>>> print(type(nom1))
<class 'str'>
>>> print(len(nom1))
6
>>> nom2='Hélène'.encode('utf8')
>>> print(nom2)
b'H\xc3\xa9l\xc3\xa8ne'
>>> print(type(nom2))
<class 'bytes'>
>>> print(len(nom2))
8
>>> nom3=nom2.decode('utf8')
>>> print(nom3)
Hélène
```

La différence de longueur entre les deux formats d'encodage provient du fait que dans une chaîne de type `bytes`, chaque caractère accentué est codé sur 2 octets.

2. Opérations élémentaires sur les chaînes

Deux chaînes peuvent être comparées avec les opérateurs `==` (égalité), `<`, `<=`, `>`, `>=` (comparaisons selon l'ordre lexicographique).
Les caractères d'une chaîne sont indicés à partir de 0. Si `nom` désigne une chaîne, le caractère d'indice i est `nom[i]`.
Il n'est pas possible de modifier isolément un caractère d'une chaîne : les chaînes de caractères sont des objets **non mutables** (contrairement aux listes).

```
>>> nom = 'Le␣corbeau'
>>> print(nom[0],nom[3],nom[5])
L c r
>>> nom[1] = 'E'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Plus généralement il est possible d'extraire une partie d'une chaîne (« slicing »), le résultat sera une chaîne. La syntaxe générale est la suivante :

`chaîne[indice début:indice fin:pas]`

Par défaut le pas est de 1. Si l'indice de début est absent cela signifie « depuis le début », si l'indice de fin est absent cela signifie « jusqu'à la fin ». Le pas peut être négatif pour un parcours de la chaîne dans l'autre sens.

```
>>> nom = 'Hélène'
>>> print nom[:3] # les 3 premiers caractères
Hél
>>> print nom[1::2] # de 2 en 2 à partir du deuxième
éèe
>>> print nom[1:] # à partir du deuxième
élène
>>> print nom[-3:] # les 3 derniers
ène
>>> print nom[4:0:-1] # du 5ième au 2ième ordre inverse
nèlé
```

Une chaîne est un objet itérable, elle peut être parcourue à l'aide d'une boucle `for` :

```
>>> for z in 'Hélène': print(z)
H
é
l
è
n
e
```

La concaténation est l'opération qui consiste à juxtaposer deux chaînes pour en faire une seule. En Python c'est l'opérateur `+` et les deux opérandes doivent être de même type :

```
>>> nom1 = "Bonjour "
>>> nom2 = "vous!"
>>> nom3 = nom1 + nom2 #concaténation
>>> nom4 = nom1 * 4    # répétition
>>> print(nom3)
Bonjour vous!
>>> print(nom4)
Bonjour Bonjour Bonjour Bonjour
>>> nom5 = nom1 + 45
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

Il est possible de convertir une valeur numérique en chaîne de caractères avec la fonction `str()` :

```
>>> nom1 = "Bonjour "
>>> print(nom1 + str(45))
Bonjour 45
```

À l'inverse, il est possible de transformer une chaîne représentant une valeur numérique en nombre avec les fonctions `int` ou `float`.
Pour connaître toutes les commandes applicables aux chaînes de caractère, taper `help(str)` ; par exemple `C.upper()` pour mettre la chaîne de caractères C en majuscules.

III Les tuples

Un tuple est similaire à une liste à deux différences près :

- Un tuple est délimité par une paire de parenthèses et non de crochets.
- Un tuple est **non mutable** (contrairement aux listes), ce qui peut être une sécurité dans certains cas. Ce type de données est également moins gourmand en ressources que les listes.

```
>>> t = (1,2,3,4,5,6)
>>> type(t)
<type 'tuple'>
>>> t[1:5]
```

```
(2,3,4,5)
>>> t[1] = 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Un tuple peut être utilisé dans une boucle for :

```
>>> for (i,j) in [(1,2),(3,4),(5,6)]:
        print(i,j)
1 2
3 4
5 6
>>> for (i,j) in enumerate(['a','b','c','d']) :
        print(i,j)
0 a
1 b
2 c
3 d
```

Conversion liste vers tuple et vice-versa :

```
>>> L1 = [1, 2, 'abs', 1.235, 'toto']
>>> L2 = tuple(L1)
>>> print(L2)
(1, 2, 'abs', 1.235, 'toto')
>>> L3 = list(L2)
>>> print(L3)
[1, 2, 'abs', 1.235, 'toto']
```

IV Les dictionnaires

1. Création d'un dictionnaire

Un dictionnaire peut être vu comme un ensemble (pas d'ordre) de couples du type :

<clé> :<valeur>

on accède à une valeur par l'intermédiaire de sa clé.
Création d'un dictionnaire :

```
>>> dico = {} # dictionnaire vide
>>> dico = { "mouse":"souris", "book" : "livre", "one":1 }
>>> type(dico)
<type 'dict'>
>>> print(dico)
{'book': 'livre', 'mouse': 'souris', 'one': 1}
>>> print(dico['mouse'])    # accès à un élément
souris
>>> print(dico['toto'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'toto'

>>> len(dico)   #nombre d'éléments
3
```

Un dictionnaire n'est pas ordonné. Les clés et les valeurs peuvent être de types divers et hétérogènes (chaînes, listes, nombres,...).

2. Opérations sur un dictionnaire

Ajout/modification d'un élément :

```
>>> dico[(1,2)] = ['un',2]
>>> print(dico)
{(1, 2): ['un', 2], 'book': 'livre', 'mouse': 'souris', 'one': 1}
>>> dico['one'] = 'un'
>>> print(dico)
{(1, 2): ['un', 2], 'book': 'livre', 'mouse': 'souris', 'one': 'un'}
```

Cet exemple montre que les dictionnaires sont **mutables**.
Suppression d'un élément :
Instruction `del` ou méthode `pop` :

```
>>> del dico[(1,2)] #supprime la valeur de clé (1,2)
>>> print(dico)
{'book': 'livre', 'mouse': 'souris', 'one': 'un'}
>>> dico.pop('book') #renvoie la valeur et supprime
'livre'
>>> print(dico)
{'mouse': 'souris', 'one': 'un'}
```

La méthode `<dict>.pop(<clé>)` renvoie la valeur correspondant et la supprime du dictionnaire.
Parcours d'un dictionnaire :
Un dictionnaire est un itérable, on peut le parcourir avec une boucle `for`, ce sont les clés qui sont parcourues :

```
>>> for c in dico :
        print(c,dico[c])
mouse souris
one un
```

V Ensembles

Un ensemble est une collection d'objets hétérogènes, distincts et non ordonnés. Un ensemble n'est **pas mutable**.

```
>>> E = {1,3,2,4,2,3} # définition par la liste de ses éléments
>>> E
set([1, 2, 3, 4])
>>> type(E)
<type 'set'>
>>> E[1] # aucun sens
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support indexing
>>> F = set([1,2,'toto']) # définition à partir d'une liste
>>> F
set([1, 2, 'toto'])
>>> len(F) # cardinal de l'ensemble
3
>>> for elt in E : # un ensemble est itérable
        print elt
1
2
3
4
>>> list(E) # conversion en liste
[1, 2, 3, 4]
>>> set() # ensemble vide
set([])
```

Opérations sur les ensembles :

```
>>> E | F # réunion
set([1, 2, 3, 4, 'toto'])
>>> E & F # intersection
```

```
set([1, 2])
>>> E == F # test d'égalité (!= pour différents)
False
>>> G = E & F
>>> G <= E # inclusion (stricte avec <, >= et > idem)
True
>>> 3 in E , 3 in F # test d'appartenance
(True, False)
```