Chapitre 1.5 : Listes et dictionnaires

I Listes

Une **liste** est un objet qui permet de stocker une suite de valeurs qui peuvent être de n'importe quel type. Les listes ressemblent beaucoup aux tuple (ce sont des séquences), mais la grande différence est qu'elles sont **mutables** : on peut modifier un élément d'une liste, on peut ajouter un élément ou retirer un élément.
La façon la plus simple (mais aussi la plus limitée) de créer une liste est de la définir en extension : on écrit les éléments 1 par 1 comme pour un tuple, mais avec des crochets au lieu des parenthèses.

```
>>> L = [1, 2, True, 'a', 0.5]
```

I.1 Accès aux éléments d'une liste

Le nombre d'éléments d'une liste `L` s'obtient grâce à `len(L)`. Les éléments de `L` sont numérotés de `0` à `len(L)-1` et on y accède grâce à `L[i]`.

```
>>> L[0]
1
>>> L[-1]
0.5
```

Si l'indice positif dépasse l'indice maximal, l'interpréteur renvoie une erreur (très classique !) `list index out of range`.
Contrairement aux chaînes de caractères, on peut modifier un élément d'une liste : les listes sont **mutables**. La syntaxe est la même que pour une affectation.

```
>>> L[2] = 3
>>> L
[1, 2, 3, 'a', 0.5]
```

I.2 Tranchage

Comme pour les chaînes de caractères, on peut utiliser le slicing pour récupérer un morceau d'une liste. Toutefois, comme une liste est mutable, il y a deux façons d'utiliser le slicing : le tranchage qui renvoie une tranche de la liste et l'affectation en tranche, qui modifie les valeurs de toute une tranche de la liste. Par exemple :

```
>>> L = [0, 1, 4, 9, 16, 25, 36]
>>> L[1:4]
[1, 4, 9]
>>> L[1:4] = [-1, -4, -9]
>>> L
[0, -1, -4, -9, 16, 25, 36]
```

I.3 Création de listes

Il existe plusieurs façons de définir une liste :

- En extension (voir précédemment).
- Par compréhension : la syntaxe générale est : `[f(x) for x in iterable if P(x)]` (comme pour les ensembles en maths).

  ```
  >>> M = [i**2 for i in range(10) if i%2 == 0]
  >>> M
  [0, 4, 16, 36, 64]
  ```

- Par conversion : on peut convertir un itérable (un range, une chaîne de caractères ou autre) en liste grâce à la fonction `list`.

  ```
  >>> list('Bonjour')
  ['B', 'o', 'n', 'j', 'o', 'u', 'r']
  >>> list(range(5))
  [0, 1, 2, 3, 4]
  ```

- Par slicing (voir précédemment).

- Les opérateurs + ou ∗ fonctionnent comme pour les tuples.

```
>>> 3*[0]
[0, 0, 0]
>>> [1, 2, 3] + [4, 5]
[1, 2, 3, 4, 5]
```

I.4 Ajout ou suppression d'un élément

On peut ajouter un élément à une liste avec la méthode `append` : si `L` est une liste et `e` un objet, alors l'instruction `L.append(e)` rajoute l'élément `e` à la fin de `L`.

```
>>> L.append(False)
>>> L
[1, 2, 3, 'a', 0.5, False]
```

Il est très fréquent d'utiliser cette méthode en partant de la liste vide `[]` pour la remplir avec une boucle.
Par exemple, le script suivant créé une liste de 1000 flottants régulièrement espacés entre 1 et 2.

```
a, b = 1, 2
n = 1000
L = []
for i in range(n):
    L.append(a + (b-a)*i/(n-1))
```

On dispose aussi de la méthode `pop` qui permet de supprimer le dernier élément d'une liste. Cette méthode renvoie aussi l'élément supprimé.

```
>>> M.pop()
64
>>> M
[0, 4, 16, 36]
```

I.5 Parcours des éléments d'une liste

Comme pour les chaînes de caractères, il existe deux façons de parcourir les éléments d'une liste :

- on fait une boucle avec un `range(len(L))`. Par exemple, pour calculer la somme des éléments de `M` :

```
s = 0
for i in range(len(M)):
    s = s + M[i]
```

- on itère directement sur les éléments de `L`. Même exemple :

```
s = 0
for element in M:
    s = s + element
```

II Le module matplotlib

Le module `matplotlib` permet de tracer des graphes de fonctions, des histogrammes, et autres... Pour l'importer on utilisera `import matplotlib.pyplot as plt`. On devra donc utiliser le préfixe `plt.` pour appeler les fonctions du module.

Pour tracer le graphe d'une fonction, on a besoin de deux listes : la liste des abscisses et la liste des ordonnées des points sur la courbe. Par exemple, pour tracer le garphe du cosinus sur $[-\pi, \pi]$:

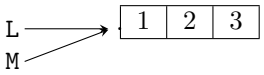
```
import matplotlib.pyplot as plt
from math import cos, pi
n = 1000
X = [-pi + 2*pi * i / (n-1) for i in range(n)] #n points entre -pi et pi
Y = [cos(x) for x in X] #ordonnées
plt.plot(X, Y) #place les points de coordonnées (X[i], Y[i])
plt.legend('cos')
plt.show() #affiche la fenetre avec le graphe
```

III Listes et copies

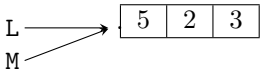
Lorsqu'on affecte une valeur à une variable, l'interpréteur python va placer la valeur dans un emplacement mémoire et retenir l'adresse de cet emplacement et l'associer au nom de la variable. Pour les listes, leur caractère **mutable** peut porter à confusion.
Lorsqu'on définit une liste : `L = [1, 2, 3]`, l'interpréteur va stocker les valeurs de la liste en mémoire et va retenir l'adresse de la première sous l'alias `L`. Considérons alors la suite d'instructions :

```
M = L
L[0] = 5
```

La variable `M` devient un nouvel alias pour l'adresse mémoire de la liste originale. On peut représenter la situation ainsi :

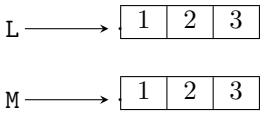


La deuxième instruction va modifier le premier élément de la liste en mémoire. Ainsi, `L[0]` et `M[0]` valent maintenant 5 !



L'instruction `M = L` n'a donc pas créé de copie de `L`. On a deux façons d'effectuer une copie de liste :

- on fait un slicing : `M = L[:]` réalise une copie en mémoire de `L`. On aurait donc :



  Les modifications effectuées sur `L` n'influent pas `M` et vice-versa.
  Attention, cette copie est superficielle : il faut prendre des pincettes avec des listes de listes ! En effet, les valeurs stockée dans la liste sont les adresses mémoires des sous-listes.
- on utilise la fonction `deepcopy` du module `copy` : `M = deepcopy(L)` réalise une copie en profondeur de `L`.

IV Listes et fonctions

Lorsqu'on appelle une fonction `f(a)` avec une variable `x`, l'interpréteur copie la valeur stockée dans `x` dans une variable locale `a`. On ne peut donc pas modifier la variable `x` dans la fonction. Par exemple

```
def reinitialise(a):
    a = 0
```

quand elle est appelée :

```
>>> x = 1
>>> reinitialise(x)
>>> x
1
```

Par contre, si `x` est mutable (par exemple une liste), la variable locale `a` contient l'adresse en mémoire du contenu de la liste et on peut aller modifier la liste en mémoire.
Pour résumer : lorsqu'une liste est passée en paramètre d'une fonction, toutes les modifications appliquées à la liste dans la fonction persistent en dehors. Par exemple la fonction :

```
def met_zero(L):
    L[0] = 0
```

quand elle est appelée avec

```
>>> M = [1, 2, 3]
>>> met_zero(M)
>>> M
[0, 2, 3]
```

V Dictionnaires

Un dictionnaire est une structure de données permettant de stocker des valeurs qui ne sont pas indexées par des entiers naturels consécutifs, mais par n'importe quelles valeurs, pas forcément d'un seul type (du moment qu'elles sont immutable). C'est donc une collection de couples (clé, valeur). En pratique, les clés sont des nombres ou des chaînes de caractères.
Pour créer un dictionnaire, la syntaxe est : `{k1 : v1, k2 : v2, ..., kn : vn}`. Le dictionnaire vide est `{}`. Le nombre d'éléments du dictionnaire s'obtient à l'aide de la fonction len.

```
>>> d = {'eleves' : 25, 'chaises' : 30, 'tables' : 29, 42 : True}
>>> len(d)
4
```

Pour accéder à la valeur associée à `clé` dans le dictionnaire `d`, on a : d[clé]. Si la clé n'est pas présente dans le dictionnaire, on obtient une erreur `KeyError`.

```
>>> d['chaises'], d[42]
30, True
```

Les dictionnaires sont **mutables** : on peut ajouter un couple, modifier une valeur ou supprimer un couple :

- pour ajouter un nouveau couple `(k, v)` au dictionnaire `d`, il suffit de l'affecter : `d[k] = v` ;
- si la clé `k` est déjà dans le dictionnaire, l'instruction précédente va modifier sa valeur ;
- pour supprimer la paire `(k ,v)`, on dispose de `d.pop(k)`, qui renvoie aussi la valeur `v`.

```
>>> d['tableau'] = 1
>>> d[42] = 42
>>> d.pop('chaises')
>>> d
{'eleves': 25, 'tables': 29, 42: 42, 'tableau': 1}
```

On peut itérer sur les clés d'un dictionnaire, et aussi sur les valeurs.

```
for cle in d:
    print(cle) #Affiche les clés du dictionnaire
for val in d.values():
    print(val) #Affiche les valeurs du dictionnaire
```

Pour copier un dictionnaire, on a la méthode `copy` (cette copie est superficielle !) ou bien la fonction `deepcopy` du module `copy`.

VI Exercices

Exercice 1 On définit `L = list(range(1, 10))`. Que font chacune des instructions suivantes ?

1. `print(L[3])`
2. `print(L[-2])`
3. `L[2] = 'b'`
4. `print(L)`
5. `L.append(10)`
6. `print(L)`
7. `d = L.pop()`
8. `print(d, L)`

■

Exercice 2

1. Écrire une fonction qui étant donné n renvoie la liste des n premières puissances de 2.
2. Écrire une instruction qui permet de définir une liste contenant les $i^2 + 1$ pour i un entier naturel et $i^2 + 1 \leq 500$.
3. Écrire une fonction `fibo(n)` qui renvoie la liste des n premiers termes de la suite de Fibonacci.
4. Écrire une fonction `nombre_zeros(L)` qui prend en argument une liste d'entiers et renvoie le nombre de

0 dans cette liste.

5. Écrire une fonction d'entête
```
def echange(L:list, i:int, j:int) -> NoneType:
```
qui échange les deux éléments d'indices `i` et `j` de la liste `L`.
6. Écrire une fonction d'entête
```
def somme(L:list) -> int:
```
qui renvoie la somme des éléments de la liste d'entiers `L`.
Comment peut-on qualifier la variable qui contient la somme ?

Exercice 3 Qu'affiche chacun des scripts suivants ?

1. .
```
L = ['b', 'a', 'c']
M = L
L[-1] = True
print(M)
M[0] = False
print(L)
N = L[:]
N[0] = 0
print(L, N)
```
2. .
```
def f(L):
    L[0] = 10
M = [1, 2, 3]
f(M)
print(M)
```
3. .
```
def g(x):
    x = x + 1
y = 1
g(y)
print(y)
```

Exercice 4
1. Écrire une fonction `met_zero2(L)` qui prend en argument une liste non vide `L` et renvoie une liste composée des mêmes éléments que `L` sauf le premier qui est remplacé par 0. Cette fonction ne doit pas modifier `L`.
2. Écrire une fonction `echange(L:list, i:int, j:int)` qui renvoie une liste avec les mêmes éléments que `L` sauf ceux d'indices `i` et `j` qui sont échangés. La liste `L` ne doit pas être modifiée.
3. Écrire une procédure `echange(L:list)` qui prend une liste de taille paire et échange les éléments d'indices pairs $2k$ avec ceux d'indice $2k+1$.

Exercice 5 On dispose d'un dictionnaire associant les noms des élèves d'une classe aux notes qu'ils ont eues à un DS.
Écrire une fonction `moyenneDS(d)` qui prend en argument un tel dictionnaire et renvoie la moyenne de la classe au DS.