Informatique Pour Tous

Cours 6 – Les listes (cours à compléter en classe)

A/ Type *list*

Un objet de type `list` est un c'est-à-dire qu'on peut faire .

C'est une liste indexée (l'ordre est important) de valeurs qui n'ont pas forcément le même .

Exemple 1:
```
>>> L = [1, 3.14, "Thé"]    # type(L) renverrait :
>>> print(type(L[0]))       # affiche le type du 1er élément, càd
>>> print(type(L[1]))       # affiche
>>> print(type(L[-1]))      # affiche type du dernier élément :
```

Exemple 2:
```
>>> L2 = [5/4,3%2,[3,4]]    # on peut déclarer tout en faisant un calcul
>>> print(type(L[0]))       # type de 5/4, càd
>>> print(type(L[1]))       # type de 3%2, càd
>>> print(type(L[2]))       # (ou, idem, L[-1]) : affiche
```

Remarque : on peut donc écrire des listes de listes ! C'est, entre autres, comme ça qu'on codera des matrices ! soit la matrice $M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$. La ligne de code permettant de déclarer *M* sous forme de liste de listes, telle qu'on accède à $M_{i,j}$ par la commande $M[i][j]$ est :

B/ Slicing

Un élément d'une liste est repéré par un . Le *slicing* permet d'extraire une .

Exemple :
```
>>> L = list(range(1,6))    # L vaut :
>>> print(L[1:3])           # affiche :
>>> print(L[4:-1])          # (idem L[4:4]) affiche :
>>> print(L[4:])            # affiche (tous à partir d'index 4 inclus) :
>>> print(L[:2])            # affiche (tous jusqu'à index 2 exclu) :
```

Si on veut `[1,2,4,5]` (on veut « dégager » l'index 2), il faut taper :

C/ Opérateurs entre les listes

- L'opérateur « + » entre 2 listes doit se lire comme la de ces 2 listes.

Exemple 1:
```
>>> L1 = [1, 2]
>>> L2 = [4, 1.45]
>>> LA = L1 + L2        # LA vaut alors :
```

- L'opérateur « * » entre une liste et un entier naturel se lit comme la de cette liste.

Exemple 2:
```
>>> LB = [1,3]*3        # LB vaut alors :
>>> N = 6
>>> LC = [0]*N          # LB vaut alors :
```

D/ Fonctions et méthodes utiles

D.1 – Quelques fonctions usuelles

- `len(L)` prend en argument une liste *L* et renvoie le nombre d'éléments dans cette liste.

Exemple 1:
```
>>> Liste = [1,5,7/3,4+2]+["hello"]
>>> print(len(Liste))      # affiche :
```

Remarque : l'index des éléments d'une liste va de .

- `list(iterable)` prend en argument un itérable (càd quelque chose dans lequel on puisse faire un *for*), et le convertit en liste, élément par élément.

Exemple 2:
```
>>> print(list("hello"))    # affiche :
>>> list(range(5,-1,-1))    # renvoie :
```

D.2 – Qu'est-ce qu'une méthode ?

Python est un langage objet. Lorsqu'on définit un objet (entre autres, une instance de classe), on peut prédéfinir des méthodes spécifiques à ce type d'objets. L'appel d'une `méthode` appliquée à un `objet` ne s'écrit pas comme une fonction. On écrira :

Exemple :
```
>>> blabla = "hello"  # la variable blabla est un objet de type
>>> blabla.upper()    # méthode spécifique aux str, renvoie
```

D.3 – Cas particulier des méthodes sur les listes

La plupart des méthodes spécifiques aux listes modifient la liste **en place** ! Cela signifie qu'elles affectent la liste (pourtant c'est une variable globale) à l'extérieur de la fonction ! Avec les types de variables que nous avions rencontrées jusqu'ici (*int, bool, float, str*), ceci était impossible. Pour les listes, il va falloir faire très attention à la portée des opérations que l'on réalise, car certaines ont une portée locale, d'autre globale… Nous y reviendrons dans un cours spécial.

Ainsi, plutôt que de choisir de **renvoyer la valeur** d'une liste modifiée, presque toutes les méthodes natives de *Python* sont **procédurales** : elles modifient la liste **en place** sans même renvoyer quoi que ce soit ! CAS SUBTILES !

Essayons de comprendre grâce à un exemple :

```
>>> L = [3, 5, 1, 4, 2]
# La fonction sorted() prend en argument une liste, et renvoie la valeur de la liste triée
>>> print(sorted(L))  # ceci affiche :
>>> print(L)          # ceci affiche :
                      # la liste a / n'a pas été modifiée ?
# La méthode .sort() trie la liste à laquelle elle s'applique et ne renvoie rien
>>> print(L.sort())   # ceci affiche :
>>> print(L)          # ceci affiche :
                      # la liste a / n'a pas été modifiée ?
```

Voyons donc une erreur très classique !

```
>>> L = [3, 5, 1]
>>> L = sorted(L)
>>> print(L)     # ceci affiche :
```

```
>>> L = [3, 5, 1]
>>> L = L.sort()
>>> print(L)     # affiche :
```

D.4 – Quelques méthodes utiles

- `L.append(elem)` permet d'ajouter la valeur de `elem` à droite de la liste `L` (dont la taille augmente de 1).

Exemple 1:

```
>>> Ltst = []
>>> Ltst = Ltst.append(3)
>>> print(Ltst) # affiche :


>>> A = []
>>> for k in range(5) :
>>>     A.append(2*k+1)
>>> print(A)     # affiche :

>>> A = []
>>> for k in range(5) :
>>>     A[k] = 2*k+1
>>> print(A)
```

- `L.insert(index, elem)` permet d'insérer la valeur de `elem` dans la liste `L`, de sorte que l'`index` après insertion soit celui imposé.

Remarque : attention à l'ordre des arguments, on pense souvent « insérer tel élément en telle position »… Les arguments sont dans l'autre sens : « insérer en telle position, tel élément ».

Exemple 2:

```
>>> Ltst = [1,2,4]
>>> Ltst.insert(0, 2)   # Ltst vaut :
>>> Ltst.insert(3, 9)   # Ltst vaut :
>>> Ltst.insert(-1, 7)  # Ltst vaut :
```

Si on veut insérer « en dernier », c'est-à-dire faire une commande similaire à `L.append(elem)`

➔ il faut taper :

Remarque : Par la suite, nous découvrirons d'autres méthodes à connaître pour les écrits, notamment `L.reverse()` (cours sur les tris) et `L.pop(index)` (cours sur les files et piles). Pour le moment, vous n'avez droit qu'à ces deux-là !

E/ Deux compléments utiles

D.1 – Test d'appartenance

Le test `a in L` renvoie `True` ssi la valeur de *a* peut être trouvée dans la liste `L`.

Exemple :

```
>>> Ltst = [1,3,7]
>>> valeur = 3
>>> print(valeur in Ltst, 4 in Ltst) # on peut l'écrire avec ou sans variable
                                     # affiche :
```

D.2 – Suppression d'un ou plusieurs éléments

La commande **del** `variable` permet normalement d'effacer le contenu d'une variable.

Remarque : attention à la syntaxe. Il ne s'agit ni d'une méthode, ni d'une fonction ! Ça ne s'écrit ni `del(variable)` ni `variable.del()` !

Exemple 1:

```
>>> x = 3
>>> del x
>>> print(x)
```

Pour les listes, on peut supprimer une partie d'une liste !

Remarque : attention, la portée est globale… ! Nous y reviendrons dans un cours dédié.

Exemple 2:

```
>>> LCar = list("Bonjour")
>>> del LCar[0]        # suppression d'un seul caractère
>>> del LCar[-1]       # attention de ne pas taper LCar = del ...
>>> print(LCar)        # affiche : ["o","n","j","o","u"]

>>> del LCar[-2:]      # on peut aussi utiliser le slicing
>>> print(LCar)        # affiche : ["o","n","j"]
```

ANNEXE : https://savoir.ensam.eu/moodle/course/view.php?id=1428