1
Les listes en python

1. Définitions

1.1 Une **liste** est un ensemble d'éléments séparés par des virgules et limité par des crochets.
`L = [7,5,3,1,5,9]`
1.2 Le nombre d'éléments d'une liste est la **longueur** de la liste.
La longueur de la liste `L` est donnée par `len(L)`.
1.3 Pour une liste de longueur N, les **indices** valides varient entre $-N$ et $(N-1)$ au sens large :

$$-N \leqslant k \leqslant N-1.$$

L'utilisation d'un indice *non* valide provoque une erreur.
`IndexError: list index out of range`
1.4 La **liste vide** est `[]`.
Elle ne contient aucun élément et sa longueur est nulle.

2. Exemples

Les éléments qui constituent une liste peuvent être de toute nature.

```
[1,2,3,4,5,6]                    # entiers
[3.14, 0.693, 2.78]              # flottants
[True, True, False]              # booléens
['a', 'bra', 'ca', 'da', 'bra']  # chaînes de caractères
[1, 1.609, 'bonjour', None]
```

3. Création d'une liste

3.1 On peut convertir un itérable en liste avec l'opérateur `list`.

```
list(range(10))
list(range(1,11))
list(range(0,30,5))
list(range(0,-10,-1))
```

3.2 On peut énumérer les éléments d'une liste si on dispose d'une formule globale.

```
[2*n+1 for n in range(10)]
[sin(3*n) for n in range(10) if sin(3*n)>0]
```

I
Indexation

4. Accès à un élément

On considère une liste `L` de longueur N.
– L'indice du premier élément est nul.
– L'indice du dernier élément est égal à $(N-1)$.
– L'élément d'indice k est `L[k]`.
– Si l'entier k est *strictement négatif*, alors $0 \leqslant N+k < N$ et la valeur de `L[k]` est égale à `L[N+k]`.

4.1 Une liste est une **variable mutable**. On peut modifier la valeur de chaque élément en lui affectant une nouvelle valeur : `L[0] = 2016`
Si une liste `L` est passée en argument à une fonction `f`, les modifications de la liste `L` effectuée à l'intérieur de la fonction `f` sont encore effectives après l'exécution de `f`.

```
def f(L):
  L[0] = 2016
  return None

L = [0,1,2,3,4,5,6]
f(L)
L
```

II
Sous-listes

5. On considère une liste `L` de longueur N et deux indices valides $i \leqslant j$.
5.1 On suppose que i et j sont *positifs*.
– La tranche `L[i:j]` est la sous-liste

$$(L_k)_{i \leqslant k < j}.$$

– La tranche `L[i:]` est la sous-liste

$$(L_k)_{i \leqslant k} = (L_k)_{i \leqslant k < N}.$$

– La tranche `L[:j]` est la sous-liste

$$(L_k)_{k < j} = (L_k)_{0 \leqslant k < j}.$$

5.2 La tranche `L[:]` est une *copie* de la liste `L` : elles ont la même taille, les mêmes éléments dans le même ordre, mais on peut modifier l'une sans changer l'autre.

```
L = [0,1,2,3,4,5,6]
M = L[:]
L == M            # True
L[3] = 0
L == M            # False
```

5.3 On suppose que les indices i et j sont *strictement négatifs*.
– La tranche `L[i:j]` est la sous-liste

$$(L_k)_{N+i \leqslant k < N+j}.$$

– La tranche `L[i:]` est la sous-liste

$$(L_k)_{N+i \leqslant k < N}.$$

– La tranche `L[:j]` est la sous-liste

$$(L_k)_{0 \leqslant k < N+j}.$$

5.4 Si $i < 0 \leqslant j$, alors la tranche `L[i:j]` est la sous-liste

$$(L_k)_{N+i \leqslant k < j}.$$

Si $j \leqslant N+i$, cette sous-liste est la liste vide.

5.5 Si $j < 0 \leqslant i$, alors la tranche `L[i:j]` est la sous-liste

$$(L_k)_{i \leqslant k < N+j}.$$

Si $i \geqslant N + j$, cette sous-liste est la liste vide.

6. Soit p, un entier non nul.
6.1 La tranche `L[i:j:p]` est la sous-liste de la tranche `L[i:j]` constituée des éléments L_k pour lesquels l'indice k vérifie

$$k = i \mod p.$$

6.2 Si p est négatif, les éléments apparaissent dans la tranche `L[i:j:p]` dans l'ordre inverse de l'ordre dans lequel ils apparaissent dans `L`.
6.3 En particulier, `L[::-1]` est une copie en miroir de `L`.

III

Opérations sur les listes

7. Recherche d'un élément
7.1 L'objet `x` est un élément de la liste `L` si, et seulement si, le booléen `x in L` est égal à `True`.
7.2 Le nombre d'occurrences de l'objet `x` dans la liste `L` est donné par `L.count(x)`.
L'élément `x` appartient à la liste `L` si, et seulement si, l'entier `L.count(x)` est strictement positif.
7.3 L'instruction `L.index(x)` retourne l'indice de la première occurrence de l'objet `x` dans la liste `L` ou une erreur `ValueError` si `x` n'est pas dans la liste `L`.

8. Insertion et suppression d'un élément
Une liste `L` peut être modifiée par les méthodes `insert`, `remove` et `pop`.
8.1 On insère l'élément `x` en position i dans la liste `L` avec l'instruction `L.insert(i, x)`. La longueur de `L` augmente d'une unité.
8.2 On enlève de `L` la première occurence de `x` avec l'instruction `L.remove(x)`. La longueur de `L` diminue d'une unité.
Si `x` n'est pas dans la liste `L`, il se produit une erreur :
`ValueError: list.remove(x): x not in list`
8.3 L'instruction `L.pop(i)` *retourne et supprime* l'élément L_i de la liste `L`. La longueur de `L` diminue d'une unité.
Par défaut, c'est le *dernier* élément de la liste qui est retourné et supprimé par la méthode `pop`.

```
from random import randint
# On lance un dé 10 fois de suite :
L = [randint(1,6) for i in range(10)]
while (L!=[]):
  x = L.pop()
  print(x, L)
```

9. Concaténation
9.1 On ajoute un élément à la fin de la liste `L` avec la méthode `append`.

```
La, Lb = [1,2,3], [4,5,6]
La.append(7)
```

9.2 On concatène deux listes avec l'opérateur `+`.

```
La + Lb
Lb = Lb + [8]
```

9.3 Dans l'exemple suivant, la méthode `extend` ajoute la liste `Lb` à la liste `La`. La liste `La` est donc modifiée.

```
La.extend(Lb)
```

9.4 L'opérateur `*` effectue des concaténations répétées.

```
3*Lb
Lb*3
```

10. Classement
10.1 Une liste peut être modifiée par la méthode `reverse`, qui inverse l'ordre dans lequel les éléments sont écrits.

```
La = [1,4,7,8,5,2]
Lb = La[::-1]
La.reverse()
La == Lb
```

10.2 Si les éléments de la liste `L` sont comparables, cette liste peut être triée par ordre croissant avec la méthode `L.sort()`.
Elle peut aussi être triée par ordre décroissant avec la méthode `L.sort(reverse=True)`.
10.3 La commande `sorted(L)` renvoie une liste triée par ordre croissant, de même longueur que la liste `L` et qui contient les mêmes éléments que `L`. Cette fois, la liste `L` n'est pas modifiée !

11. Opérations algébriques
11.1 On suppose que les éléments de la liste `L` sont comparables : nombres ou chaînes de caractères.
Le plus petit élément de la liste `L` est donné par `min(L)`.
Le plus grand élément de la liste `L` est donné par `max(L)`.
11.2 Si les éléments de la liste `L` sont des *nombres*, on obtient la somme de ces nombres avec l'instruction `sum(L)`.