IngeSUP - Cours 04 - Listes et chaînes I

Sommaire

- Les listes en python
  - Création d'une liste
  - Accéder à un élément d'une liste par rang positif
  - Accéder à un élément d'une liste par rang négatif
  - Manipuler des listes
    - Ajouter des éléments en fin de liste avec append()
    - Ajouter plusiuers éléments à la fois en fin de liste avec extend()
    - Ajouter des éléments à la place voulue avec insert()
    - Modifier la valeur d'une liste grâce à son rang
    - Supprimer des éléments de la liste
  - Opérateurs sur les listes
  - Autres fonctions prédéfinies en python sur les listes
  - Parcourir les listes
- Les chaînes en python
  - Caractères et chaînes
  - Accéder à un caractère d'une chaine par rang
  - Opérations sur les chaînes
  - Parcourir les chaînes
- Exercices de TD

Les listes en python

Une liste est une suite d'éléments regroupés. Cela peut être une liste d'entiers, par exemple $[5, -7, 12, 99]$, ou bien une liste de chaînes de caractères, par exemple $[" Mars ", " Avril ", " Mai "]$ ou bien des objets qui peuvent être de différents types $[3.14, " pi ", 10e-3, " x ", True]$.

> **Note**
>
> Une liste commence par des crochets et finit par des crochets. Les éléments d'une liste sont séparés par des virgules. Parfois on les appelle des **tableaux**. En ce qui concerne Python ces deux mots sont synonymes.

D'un point de vue **algorithmique** listes sont des **structures de données**. Une structure de donnée est un objet dans un programme qui contient une série de données.

Une structure de données simple peut être par exemple un tableau qui contient les composantes d'un vecteur ou une liste de noms.

Création d'une liste

Les listes en Python peuvent être créées en plaçant simplement la séquence entre crochets[].

Une liste peut contenir des valeurs similaires dans plusieurs endroits.

Entrée [ ]:

```
# Liste vide
liste = []

# liste avec éléments
semaine = ['lundi','mardi']

# liste avec éléments de types différents
liste2 = ['ESME','Sudria', 3, 2021]

# liste entiers
nombres = [1, 2, 3]
```

Autres exemples

Entrée [ ]:

```
lab_group0 = ["Théo", "Emilie", "Sarah", "Marc"]

print(lab_group0)

valeurs = [4.3, 5.2, 0.75, 3.14]

print(valeurs)
```

En bref

Une liste se définit par des éléments entre crochets :

- liste1 = $[5, 4, 3, 2, 1]$, une liste de 5 entiers,
- liste2 = $[" \mathit{Vendredi} ", " \mathit{Samedi} ", " \mathit{Dimanche} "]$, une liste de 3 chaînes de caractères,
- liste3 = [], **la liste vide** (très utile pour la compléter plus tard !).

Accéder à un élément d'une liste par rang positif

Pour obtenir un élément de la liste, il suffit d'écrire ***liste[i]*** où *i* est le **rang** de l'élément souhaité.

> Attention ! Le piège c'est que l'on commence à compter à partir du rang 0 !

Par exemple après l'instruction $liste = [" A "," B "," C "," D "," E "," F "]$

- liste[0] renvoie "A"
- liste[1] renvoie "B"
- liste[2] renvoie "C"
- liste[3] renvoie "D"
- liste[4] renvoie "E"
- liste[5] renvoie "F"

	"A"	"B"	"C"	"D"	"E"	"F"
rang :	0	1	2	3	4	5

Entrée [ ]:

```
liste = [1,2,3]
print("liste au rang 0", liste[0])
print("liste au rang 1", liste[1])
print("liste au rang 2", liste[2])
```

> 🚧 **Questions de cours**
>
> Que se passe t-il si on essaie d'afficher une valeur au rang 3 ? Essayez pour voir !

Accéder à un élément d'une liste par rang négatif

Python est l'un des rares langages de programmation qui permet d'accéder aux éléments d'une liste à l'aide d'indices négatifs qui commencent par `-1` en partant du dernier élément.

Ainsi, `-1` est associé au dernier élément de la liste et `-taille` au premier élément de la liste en utilisant les indices négatifs.

	"A"	"B"	"C"	"D"	"E"	"F"
rang :	-6	-5	-4	-3	-2	-1

> 💡 **Astuce**
>
> **Pour accéder au dernier élément d'une liste (quel que soit son rang) :**
>
> On peut utiliser le rang ***-1***. La valeur ***-1*** mise entre crochets permet d'accéder au **dernier élément de la liste.**

Entrée [ ]:

```
liste = ["one", "two", "three", "four"]
print(liste[-1])
```

Manipuler des listes

Ajouter des éléments en fin de liste avec `append()`

Des éléments peuvent être ajoutés à la liste à l'aide de la fonction `append()` . Un seul élément à la fois peut être ajouté avec cette méthode.

Pour ajouter plusieurs éléments à l'aide de la méthode `append()` , **il faudra faire appel aux boucles**.

Entrée [ ]:

```
# creation de la liste
liste = []
# ajouter des éléments à la liste
liste.append(1)
liste.append(2)
liste.append(3)

print("\n Liste après l'ajout de trois éléments ")
print(liste)
```

Entrée [ ]:

```
liste = []
# Avec une boucle
for i in range(1, 4):
    liste.append(i)
print(liste)
```

> 🚧 **Questions de cours**
>
> Si on avait mis le `print(liste)` **à l'intérieur** de la boucle **for**, quel code aurait été affiché ? Pourquoi ?

Ajouter plusiuers éléments à la fois en fin de liste avec `extend()`

Plusieurs éléments peuvent être ajoutés à la fois en fin de liste à l'aide de la fonction `extend()` .

Entrée [1]:

```
liste1 = [1,2]
liste2 = [3,4]

liste1.extend(liste2)  # Place les éléments de la liste « liste2 » à la fin de la liste « l
print(liste1)          # Affiche : [1, 2, 3, 4]
```

```
[1, 2, 3, 4]
```

Ajouter des éléments à la place voulue avec `insert()`

Pour l'ajout d'élément à la position souhaitée, la méthode `insert()` est utilisée.

Contrairement à `append()` qui prend un seul argument, la méthode `insert()` nécessite deux arguments (position et valeur).

Entrée [3]:

```
ma_liste = [0,1,2,4]

# ajouter l'élément 12 à la position 0
ma_liste.insert(0, 12)
# [12, 0, 1, 2, 4]

# ajouter l'élément 12 à la position 2
ma_liste.insert(2, 12)
# [12, 0, 12, 1, 2, 4]

print(ma_liste)
```

```
[12, 0, 12, 1, 2, 4]
```

Modifier la valeur d'une liste grâce à son rang

La notation avec les crochets de type $liste[i]$, en plus de nous donner la **valeur stockée dans la liste au rang i**, peut s'utiliser comme une variable.

Grâce à l'affectation (=) **on peut directement modifier la valeur stockée au rang i**.

Entrée [ ]:

```
liste = ["e","p","s"]
print(liste[2])    # Affiche s

liste[2] = "z"    # On change le 3ème élément
print(liste)

liste[0] = "f"    # Cette fois on change le 1er
print(liste)
```

Supprimer des éléments de la liste

Supprimer un élément en utilisant sa valeur

Les éléments peuvent être supprimés de la liste à l'aide de la fonction `remove()` , mais une erreur se produit si l'élément n'existe pas dans la liste.

La méthode `remove()` ne supprime qu'un élément à la fois, le premier élément trouvé.

Pour supprimer plusieurs éléments, on doit utiliser une boucle.

Entrée [ ]:

```
liste = [5, 7, 8, 12]

liste.remove(5) # Supprimer de la liste la première valeur 5 trouvée
liste.remove(8) # Supprimer de la liste la première valeur 8 trouvée
print("\nListe après la suppression: ")
print(liste)
```

Supprimer un élément en utilisant son rang

Pour supprimer un élément mentionné à l'aide du nom de la liste et son indice (rang), on utilise le mot-clé **del** en mentionnant le nom de la liste et le rang de l'élément qu'on veut supprimer.

> ⚠ **Attention**
>
> Attention **del** n'est pas une fonction mais un mot clé de python. Il n'y a donc pas d'utilisation des parenthèses !

Entrée [ ]:

```
# syntaxe del liste[indice]
Liste = [10, 2, 3, 9, 2, 1, 2, 3, 2]
del Liste[2]
# -> Liste = [10, 2, 9, 2, 1, 2, 3, 2]
print(Liste)
```

Vider une liste

Pour vider une liste il suffit de l'écraser par une liste vide.

Entrée [13]:

```
Liste = [10, 2, 3, 9, 2, 1, 2, 3, 2]
print(liste)

Liste = []
print(liste)
```

```
[10, 2, 3, 9, 2, 1, 2, 3, 2]
[10, 2, 3, 9, 2, 1, 2, 3, 2]
```

Opérateurs sur les listes

L'opérateur *

On peut utiliser l'opérateur * avec une liste et un nombre entier **pour répéter le même élément dans la liste**.

Entrée [ ]:

```
ma_liste = ["Thomas"]*5
print(ma_liste)

ma_liste_de_zeros = [0]*25
print(ma_liste_de_zeros)

L1 = [1, 2, 3]
L2=L1 * 3
print(L2)
# [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

L'opérateur +

On peut utiliser l'opérateur + avec deux listes **pour fusionner la deuxième à la premiere**.

Entrée [ ]:

```
liste = [1,2,3] + [4,5,6]
print(liste)

liste1 = [1,2,3]
liste2 = [4,5,6]
liste = liste1 + liste2
print(liste)
```

L'opérateur `in`

Cet opérateur est utilisé pour vérifier si un élément est présent ou non dans la liste. Il renvoie ***True*** si l'élément est présent dans la liste, sinon il renvoie ***False***.

Entrée [ ]:

```
Liste = [10, 2, 3, 9, 2, 1, 2, 3, 2]
if 9 in Liste:
    print("élément 9 est présent dans la liste")
else:
    print("l'élément 9 n'est pas présent dans la liste")
```

L'opérateur `not in`

Cet opérateur est utilisé pour vérifier si un élément n'est pas présent dans la liste. Renvoie ***True*** si l'élément n'est pas présent dans la liste, sinon renvoie ***False***.

Entrée [ ]:

```
List = [10, 2, 3, 9, 2, 1, 2, 3, 2]
if 9 not in List:
    print("l'élément n'est pas présent dans la liste")
else:
    print("l'élément est présent dans la liste")
```

Autres fonctions prédéfinies en python sur les listes

💡 La fonction `len`

Pour compter le nombre d'éléments d'une liste on utilise la fonction pédefinie `len()`

Entrée [ ]:

```
# syntaxe : len(liste)
liste = [10, 2, 3, 9, 2, 1, 2, 3, 2]
print(len(liste))
# -> 9
```

> **Note**
>
> Les éléments d'une liste de longueur len(l) sont **indicés (numérotés en terme de rang) de 0 à len(l) -1**.

La fonction `sum`

Pour faire la somme de tous les éléments d'une liste on utilise la fonction `sum()`

Entrée [ ]:

```
# syntaxe : sum(List)
liste = [1, 6, 3, 9]
print(sum(liste))
```

La fonction `count`

Pour calculer le nombre d'occurrence d'un élément donné de la liste on utilise la fonction `count()` .

Entrée [ ]:

```
# syntaxe : List.count(valeur)
liste = [10, 2, 3, 9, 2, 1, 2, 3, 2]
print(liste.count(3))
# -> 2
```

La fonction `index`

La fonction `index` retourne le premier rang de la valeur passée en paramètres.

Entrée [ ]:

```
#syntaxe : List.index(valeur)
liste = [10, 2, 3, 9, 2, 1, 2, 3, 2]
print(liste.index(2))
# -> 1
```

La fonction `sort`

La fonction `sort` modifie la liste en triant ses valeurs.

- **Trier les éléments de la liste par ordre croissant :**

Entrée [14]:

```
liste = [7, 9, 1, 3, 0, 5]

liste.sort()      # Trie la liste en la modifiant.
print(liste)      # Affiche : [0, 1, 3, 5, 7, 9]
```

```
[0, 1, 3, 5, 7, 9]
```

- **Trier les éléments de la liste par ordre décroissant :**

Entrée [15]:

```
liste = [7, 9, 1, 3, 0, 5]

liste.sort(reverse=True)  # Trie la liste en la modifiant.
print(liste)              # Affiche : [9, 7, 5, 3, 1, 0]
```

```
[9, 7, 5, 3, 1, 0]
```

💡 La fonction `sorted`

La fonction `sort` est une fonction externe qui ne modifie pas la liste en triant ses valeurs. Elle crée une nouvelle liste trée.

- **Trier les éléments de la liste par ordre croissant :**

Entrée [16]:

```
liste1 = [7, 9, 1, 3, 0, 5]

liste2 = sorted(liste1)     # Renvoie une liste triée sans modifier la liste d'origine « l
print(liste1)      # Affiche : [7, 9, 1, 3, 0, 5]
print(liste2)      # Affiche : [0, 1, 3, 5, 7, 9]
```

```
[7, 9, 1, 3, 0, 5]
[0, 1, 3, 5, 7, 9]
```

- **Trier les éléments de la liste par ordre décroissant :**

Entrée [17]:

```
liste1 = [7, 9, 1, 3, 0, 5]

liste2 = sorted(liste1, reverse=True)     # Renvoie une liste triée sans modifier la liste
print(liste1)      # Affiche : [7, 9, 1, 3, 0, 5]
print(liste2)      # Affiche : [9, 7, 5, 3, 1, 0]
```

```
[7, 9, 1, 3, 0, 5]
[9, 7, 5, 3, 1, 0]
```

💡 La fonction `reverse`

La fonction reverse modifie la liste en inversant l'ordre de ses valeurs.

Entrée [20]:

```
liste1 = [7, 9, 1, 3, 0, 5]

liste1.reverse()   # Inverse l'ordre des éléments de la liste en la modifiant.
print(liste1)      # Affiche : [5, 0, 3, 1, 9, 7]
```

```
[5, 0, 3, 1, 9, 7]
```

💡 La fonction `reversed`

La fonction reversed est une fonction externe qui ne modifie pas la liste en inversant l'ordre de ses valeurs. Elle crée une nouvelle liste inversée.

Entrée [19]:

```
liste1 = [7, 9, 1, 3, 0, 5]

liste2 = list(reversed(liste1));    # Renvoie une liste inversée sans modifier celle d'orig
print(liste1)      # Affiche : [7, 9, 1, 3, 0, 5]
print(liste2)      # Affiche : [5, 0, 3, 1, 9, 7]
```

```
[7, 9, 1, 3, 0, 5]
[5, 0, 3, 1, 9, 7]
```

Parcourir les listes

Parcourir les rangs de la liste avec la boucle `for...range`

On a vu que dans les listes chaque valeur est sotckée dans une case rangée, qu'on appelle aussi un indice et dont la valeur démarre à 0 (pour le tout premier élément de la liste).

> **Note**
>
> A partir du rang d'un élément on peut obtenir la valeur d'un élément en utilisant **les crochets**.

En effet pour un indice i donné appartenant à l'ensemble des indices de la liste nommée $liste$:

- ***i*** désigne l'indice (le **rang**).
- ***liste[i]*** désigne la **valeur** stockée à cet indice.

> **Astuce**
>
> Si on a besoin de ces deux informations à la fois (le rang et la valeur), on utilisera la boucle `for...range` pour parcourir la liste.

Voici une manière de faire :

Entrée [ ]:

```
print(liste)
n = len(liste)
for i in range(n):
    # Le rang : la valeur
    print(i, ":", liste[i])
```

Cette boucle affiche le rang et la valeur de tous les éléments de la liste séparés par les deux points (:)

Parcourir les rangs de la liste avec la boucle `while`

Entrée [3]:

```
liste = [10, 2, 3, 9, 2, 1, 2, 3, 2]
n = len(liste)
i = 0
while i < n:
    # Le rang : la valeur
    print(i, ":", liste[i])
    # On passe au rang suivant
    i += 1
```

```
0 : 10
1 : 2
2 : 3
3 : 9
4 : 2
5 : 1
6 : 2
7 : 3
8 : 2
```

Parcourir les valeurs de la liste avec la boucle `for...in`

Lorsqu'on veut parcourir une liste (ou une chaîne de caractères) on peut se demander s'il est toujours utile d'accéder à l'indice des éléments...

Si ça n'est pas le cas on peut utiliser la boucle for pour **boucler directement sur les valeurs** de la liste et pas sur les indices, voici comment faire:

```
for element in liste:
    print(element)
```

Ici *element* est une variable dont le nom est choisi par la personne qui programme.

Entrée [ ]:

```
liste = [1, 10, 100, 250, 500]

for v in liste:
    print(v, end=" ")

# Pas de recours à range(len(...))
# On affiche directement les valeurs. Pas besoin d'indices !
# 1 10 100 250 500
```

Parcourir les indices et les valeurs en même temps avec la boucle `for...enumerate`

Si nous souhaitons itérer (boucler) sur les valeurs d'une liste et connaitre également la position de ces valeurs dans la liste, nous pouvons utiliser la fonction `enumerate` :

Voici comment elle fonctionne :

```
    for indice, valeur in enumerate(liste):
        print(indice, valeur)
```

Ici indice et valeur sont des variables dont le nom est choisi par la personne qui programme.

Entrée [ ]:

```
lab_group0 = ["Etienne", "Emilie", "Jean", "Rachid", "Brian"]

for n, member in enumerate(lab_group0):
    print(n, member)
```

Les chaînes en python

Caractères et chaînes

Un caractère est un symbole unique, par exemple une lettre minuscule "a", une lettre majuscule "B", un symbole spécial "&", un symbole représentant un chiffre "7", une espace " ", que l'on notera aussi ' '.

Pour désigner un caractère, il faut le mettre entre guillemets simples 'z' ou entre guillemets doubles "z".

Une chaîne de caractères est une suite de caractères, comme un mot "Bonjour", une phrase 'Il fait beau.', un mot de passe "N[w5ms}e!". Le type d'un caractère ou d'une chaîne est **str** (pour string).

Entrée [5]:

```
chaine = "Bonjour"

print(type(chaine))
```

```
<class 'str'>
```

Accéder à un caractère d'une chaine par rang

Les chaines de caractères se comportent comme des listes de caractères.

Ainsi pour obtenir un caractère de la chaine, il suffit d'écrire ***chaine[i]*** où *i* est le **rang** du caractère souhaité.

Comme pour les listes on peut accéder aux caractères d'une chaine à l'aide d'indices négatifs qui commencent par `-1` en partant du dernier élément.

Ainsi, `-1` est associé au dernier caractère de la chaine et `-taille` au premier caractère de la chaine en utilisant les indices négatifs.

Par exemple pour l'instruction $chaine =$" $Hello$ " on a :

Lettre :	H	e	l	l	o
rang positif :	0	1	2	3	4
rang négatif :	-5	-4	-3	-2	-1

- chaine[0] et chaine[-5] renvoie "H"
- chaine[1] et chaine[-4] renvoie "e"
- chaine[2] et chaine[-3] renvoie "l"
- chaine[3] et chaine[-2] renvoie "l"
- chaine[4] et chaine[-1] renvoie "o"

Opérations sur les chaînes

L'opérateur *

On peut utiliser l'opérateur * avec une chaine de caractères et un nombre entier **pour répéter la même chaine un certain nombre de fois**.

Entrée [7]:

```
chaine = "AB"*5
print(chaine)   # Affiche : "ABABABABAB"

chaine1 = "Bonjour"
chaine2 = chaine1 * 3
print(chaine2)  # Affiche : "BonjourBonjourBonjour"
```

```
ABABABABAB
BonjourBonjourBonjour
```

L'opérateur +

On peut utiliser l'opérateur + avec deux chaines de caractères **pour concaténer la deuxième à la premiere**.

La concaténation signifie la mise bout à bout de deux chaînes.

Par exemple "para"+"pluie" donne la chaîne "parapluie".

Entrée [8]:

```
chaine = "Bonjour" + " tout le monde."
print(chaine)

chaine1 = "Bonjour"
chaine2 = " tout le monde."
chaine = chaine1 + chaine2
print(chaine)
```

```
Bonjour tout le monde.
Bonjour tout le monde.
```

La chaîne vide "" est utile lorsque l'on veut initialiser une chaîne avant d'y ajouter d'autres caractères.

L'opérateur `in`

Cet opérateur est utilisé pour vérifier si un caractère ou une chaine de caractères est présent(e) ou non dans la chaine. Il renvoie ***True*** si le caractère ou la chaine de caractères est présent(e) dans la chaine, sinon il renvoie ***False***.

Entrée [10]:

```
chaine = "Bonjour"

if "B" in chaine:
    print("Le caractère B est présent dans la chaine Bonjour")
else:
    print("Le caractère B n'est pas présent dans la chaine Bonjour")

if "jour" in chaine:
    print("La chaine jour est présente dans la chaine Bonjour")
else:
    print("La chaine jour n'est pas présente dans la chaine Bonjour")
```

```
Le caractère B est présent dans la chaine Bonjour
La chaine jour est présente dans la chaine Bonjour
```

L'opérateur `not in`

Cet opérateur est utilisé pour vérifier si un caractère ou une chaine de caractères n'est pas présent(e) ou non dans la chaine. Il renvoie ***True*** si le caractère ou la chaine de caractères n'est pas présent(e) dans la chaine, sinon il renvoie ***False***.

Entrée [11]:

```
chaine = "Bonjour"

if "B" not in chaine:
    print("Le caractère B n'est pas présent dans la chaine Bonjour")
else:
    print("Le caractère B est présent dans la chaine Bonjour")

if "jour" not in chaine:
    print("La chaine jour n'est pas présente dans la chaine Bonjour")
else:
    print("La chaine jour est présente dans la chaine Bonjour")
```

```
Le caractère B est présent dans la chaine Bonjour
La chaine jour est présente dans la chaine Bonjour
```

Parcourir les chaînes

La longueur d'une chaîne est le nombre de caractères qu'elle contient. Elle s'obtient, comme avec les listes, par la fonction `len()` . Par exemple len("Hello World") renvoie 11 (l'espace compte comme un caractère).

Entrée [12]:

```
chaine = "Hello World"
n = len(chaine)
print(n)
```

```
11
```

Parcourir les rangs de la chaine avec la boucle `for...range`

On a vu que dans les chaine chaque valeur est sotckée dans une case rangée, qu'on appelle aussi un indice et dont la valeur démarre à 0 (pour la toute première lettre de la chaine).

Entrée [4]:

```
chaine = "Bonjour"
n = len(chaine)
for i in range(n):
    # Le rang : la lettre
    print(i, ":", chaine[i])
```

```
0 : B
1 : o
2 : n
3 : j
4 : o
5 : u
6 : r
```

Parcourir les lettre de la chaine avec la boucle `for...in`

La boucle `for...in` est utilisée pour les parcours séquentiels. On a vu qu'elle affichait les valeurs d'une liste mais elle peut aussi servir **pour parcourir une chaîne de caractères**.

```
for element in sequence:
    instructions
```

Ici element est une variable choisie par la personne qui code.

Entrée [ ]:

```
chaine = "hello world!"
for lettre in chaine:
    print(lettre)
```

Exercices de TD

Vous pouvez maintenant vous exercer à partir du notebook [TD 04 - Listes et chaînes I (../TD/TD%2004%20-%20Listes%20et%20cha%C3%AEnes%201.ipynb)](../TD/TD%2004%20-%20Listes%20et%20cha%C3%AEnes%201.ipynb).