I_{02} Manipulation des listes et chaines

PCSI 2020 – 2021

I Liste

Une liste est un objet de python qui peut contenir plusieurs objets (on parle d'objet itérable).
D'autres exemples de tels objets sont les tuples et les chaines de caractères.

1. Création d'une liste

On peut créer une liste de plusieurs manières :

```
L = [1,2,'a',[23,1]] #création d'une liste dont on connait les éléments
L = [] # Liste vide.  Utile si on veut la remplir progressivement.
L = [0] * 5 # [0,0,0,0,0] # Utile si on connait à l'avance la taille de la liste
```

On peut aussi remplir une liste en utilisant la fonction `list` qui prend en argument un objet itérable. Exemple :

```
L = list("bonjour") # L contient ['b', 'o', 'n', 'j', 'o', 'u', 'r']
```

Liste en compréhension[1] :

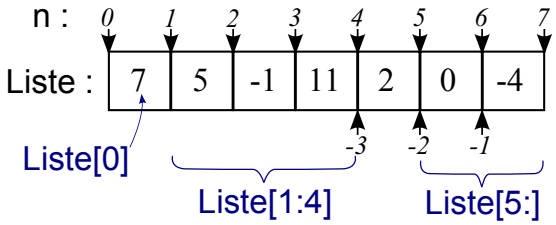
```
L = [i**2 for i in range(4)] # L contient [0, 1, 4, 9]
L = [f(x) for x in iterable] # [f(iterable[0]), f(iterable[1]), ...]
L = [i**2 for i in range(4) if i**3>1] # L contient [4, 9]
```

2. Accès à un élément

Le nombre d'éléments peut s'obtenir grâce à la fonction len.
Par exemple `len([1,1,1])` renvoie 3.

⚠ Les éléments d'une liste L sont indexés de 0 à len(L)-1.

L'élément de la liste d'indice i s'obtient par $L[i]$.
Si on utilise un indice i « négatif », python interprète cela comme « on recule en partant de la fin ». Ainsi, si i est négatif, $L[i]$ est équivalent à $L[len(L)+i]$



Exemple : si `L = [12.5, 'a', 7, [4,3],-8]`

1. $L[2]$ vaut 7
2. $L[3]$ vaut [4,3]
4. Donner un deuxième moyen d'accéder $L[2]$: $L[-3]$
5. $L[-len(L)]$ vaut : $L[0] = 12.5$

3. Extraction d'une sous-liste

La syntaxe $L[deb : fin]$ permet de récupérer la sous-liste constituée des éléments de L allant de deb inclus à fin exclu.
On peut omettre l'un des deux et dans ce cas là python va « jusqu'au bout ». Par exemple $L[: fin]$ est équivalent à $L[0 : fin]$ et $L[deb :]$ est équivalent à $L[deb : len(L)]$ (et non pas $len(L)-1$ auquel cas on ne prendrait pas le dernier élément)

1. façon avancée (et rapide) de créer une liste en la remplissant, je ne l'exige pas de vous, mais si vous le connaissez vous pouvez l'utiliser.

Exemple : si `L = [12.5, 'a', 7, [4,3],-8]`

1. $L[1:3]$ vaut $['a', 7]$
2. $L[1:2]$ vaut $['a']$
3. $L[2:-1]$ vaut $[7, [4,3]]$
4. $L[:2]$ vaut $[12.5,'a']$
5. $L[:]$ vaut $[12.5,'a', 7, [4,3], -8]$
6. $L[2:2]$ vaut $[]$

Il est aussi possible de rajouter le « pas » avec lequel on veut se déplacer (qui vaut par défaut 1) en utilisant la syntaxe : $L[deb : fin : pas]$.

Les éléments seront alors `[L[deb], L[deb+pas], L[deb+2pas], ..., L[deb+k*pas]]` tel que $deb + k \times pas < fin$ et $deb + (k+1) \times pas \geq fin$.

Exemple : si `L = [1,11,2,22,3,33,4,44]`

1. $L[1:6:2]$ vaut $[11,22,33]$
2. $L[1:5:2]$ vaut $[11,22]$
3. $L[::2]$ vaut $[1,2,3,4]$
4. $L[::3]$ vaut $[1,22,4]$
5. $L[:2:-1]$ vaut $[44,4,33,3,22]$
6. $L[3:4:-1]$ vaut $[]$

4. Modification d'un élément

Les listes en python sont des objets qui sont dit mutables , c'est-à-dire que l'on peut modifier un élément de la liste « en place » (sans être obligé de recopier la liste). Par contre, les tuples et les chaines de caractères ne sont pas mutables [2].
Exemple :

```
L = [1, 11, 2, 22, 3, 33, 4, 44]
L[2] = -10
print(L) # L contient maintenant [1, 11, -10, 22, 3, 33, 4, 44]
t = (1,2,3)
t[1] = 5 # erreur : 'tuple' object does not support item assignment
```

5. Concaténation

L'opération de mise bout à bout de deux liste s'appelle la concaténation.
En python, elle se fait à l'aide de +.

```
L = [1, 11, 2] + [22, 3, 33, 4, 44]
# L contient  [1, 11, 2, 22, 3, 33, 4, 44]
L = [22, 3, 33, 4, 44] + [1, 11, 2]
# L contient  [22, 3, 33, 4, 44, 1, 11, 2,]
```

Attention, la concaténation a l'air d'une opération simple, mais il faut avoir conscience qu'il y a un important travail pour le processeur et la mémoire.

6. Ajout d'un élément

Python est un langage orienté « objet ». Un « objet » en informatique est une structure qui dispose d'attributs et de méthodes. Une méthode est en quelque sorte une fonction qui va s'appliquer à l'objet auquel elle « appartient » et la syntaxe pour utiliser une méthode est `objet.methode(arg1,arg2,...)`.
Pour ajouter un élément à la fin d'une liste, nous allons utiliser la méthode append de la classe liste qui prend en argument l'élément que l'on souhaite ajouter.

2. Il s'agit ici d'une spécificité du langage et non pas d'une généralité en algorithmique.

```
L = [1, 2]
L.append(3) # L est modifiée et vaut maintenant [1, 2 , 3]
```

7. Suppression d'un élément

La suppression de l'élément i se fait à l'aide de $del(L[i])$.

```
L=[1,2,3,1,2,3]
del(L[4])
#L contient [1, 2, 3, 1, 3]
```

8. Comportement lors d'une copie ou d'un appel de fonction

Lorsque l'on écrit $L2 = L1$, $L2$ ne contient pas une copie de la liste $L1$ mais pointe vers la même liste.

```
L1 = [1,2,3,1,2,3]
L2= L1 # L2 pointe vers la même liste que L1
L1[ 3 ] = 5 # on modifie la liste vers laquelle pointe L1
print(L2) # [1,2,3,5,2,3] la liste vers laquelle pointe L2 est aussi
   modifiée puisque c'est la même !
L2 = L1[:] # on crée une nouvelle liste qui est la copie de L1
L1[ 3 ] = 0
print(L2) # non modifié [1,2,3,5,2,3]
```

Lien Python tutor

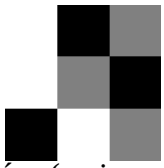
Lors du passage en argument d'une liste à une fonction, le comportement est le même.
Ainsi, avec une fonction, une liste se comporte un peu comme une variable globale : ATTENTION !
Lien Python tutor

```
L = [1,2,3,1,2,3]
def f(liste):
    liste[0] = -10
    return None
f(L)
print(L) #[-10,2,3,1,2,3]
```

9. Tableau à plusieurs dimensions

Pour faire un tableau à plusieurs dimensions on peut créer une liste de liste. Par exemple pour une image de trois pixels sur trois on peut coder le niveau de gris de la façon suivante :

```
image = [[1,0,0.5 ],[1,0.5,0 ],[0,1,0.5 ]]
# 1 = blanc, 0 = noir, 0.5 = gris
# L[0][1] 1ère ligne , 2e colonne. Couleur noir.
```



Nous verrons plus tard dans l'année des moyens plus efficace de manipuler ce type de données (mais on pourra plus difficilement modifier la taille).

II Chaines de caractères

Une chaine de caractères est un objet de python qui contient du texte. Un texte étant constitué de plusieurs caractères, c'est un objet itérable.

1. Création d'une chaine

On peut créer une chaine de plusieurs manières. Lorsque l'on connait le texte que l'on veut écrire, pour qu'il soit interprété comme du texte et non comme un nom de variable, il suffit de l'encadrer de guillemets ' ou " ou bien de triples guillemets ''' ou """ si le texte est sur plusieurs lignes.

```
texte_vide = ''
texte = 'bonjour'
texte2 = "bonjour"
texte3 = """ Ceci est
un texte sur
plusieurs lignes"""
```

On peut aussi initialiser une chaine de caractère en utilisant la fonction `str` qui prend en argument un objet et qui tente de le convertir en texte.

```
a = str(23) # a contient '23'
a = str(5*6)# '30'
a = str(5/3)# '1.6666666666666667'
```

Si le texte comporte des guillemets simples ' alors il faut l'encadrer de guillemets doubles " et réciproquement sous peine d'avoir de mauvaises surprises. Par exemple :

- `'j'aime la physique'` doit être écrit `"j'aime␣la␣physique"`
- `"Mme␣Didry␣:␣"Moi aussi !"␣"` ⇒ `'Mme␣Didry␣:␣"␣Moi␣aussi␣!"␣'`

2. Accès à un élément, extraction d'une partie.

L'indexation est exactement la même que pour les listes. Le nombre de lettre dans la chaine peut s'obtenir grâce à la fonction len. Les éléments de la chaine sont indexés de 0 à len(L)-1.

Exemple : si `t = 'Youpie␣!!!'`

1. `t[1]` vaut 'o'
2. `t[-4]` vaut ' '
3. `t[:6]` vaut 'Youpie'
5. `t[::-1]` vaut '!!! eipuoY'
6. `t[1:1]` vaut ''

3. Modification d'un élément

En python, contrairement aux listes, les chaines ne sont pas mutables.
Il n'est donc pas possible de modifier une chaine de caractère [3].

```
c = 'Phisique␣!' #oups, je me suis trompé
c[2] = 'y' # erreur : 'str' object does not support item assignment
```

4. Concaténation

De même que pour les listes, la concaténation se fait à l'aide de +.

```
s = 'bonjour'+"␣j'aime␣" + 'la␣physique'
#s contient "bonjour j'aime la physique"
```

Proposer une méthode pour corriger la faute d'orthographe du paragraphe précédent (`c = 'Phisique␣!'`).
c = c[:2] + 'y' + c[3 :]
Pour ajouter un élément à la fin d'une chaine de caractère, on fait la concaténation avec ce nouvel élément et on affecte le résultat à la chaine de caractère précédente (ou à une nouvelle) :

3. En particulier, il n'y aura pas de fonction d'ajout d'un élément à la fin comme pour les listes, ni pour supprimer un élément

```
s = "bonjour␣tu␣aimes␣la␣physique␣"
s = s + '?' #s contient "bonjour tu aimes la physique ?"
```

5. Caractères particuliers

Deux caractères particuliers doivent être connus :

- La tabulation (large espace) se note '$\backslash t$' `print('a\tb')` affiche a b
- Le retour à la ligne se note '$\backslash n$'
  Par exemple `print('a\nb')` écrira sur une première ligne a, puis passera à la ligne et écrira b en dessous.

6. Quelques méthodes utiles

Les chaines de caractères disposent d'un grand nombre de méthodes. Par exemple `'aaa'.upper()` renvoie 'AAA'. Ces méthodes ne sont pas à connaitre, mais deux d'entre elles pourront être utile lorsque nous essayeront de lire/écrire dans des fichiers.

6.a. Transformer une chaine de caractères en listes de chaines de caractères

La méthode split permet de couper une chaine de caractère en plusieurs chaines de caractères. Par exemple, si on veut récupérer les mots d'une phrase, on peut vouloir couper lorsque l'on rencontre un espace.
Le caractère qui doit jouer le rôle de séparateur est l'argument passé à la méthode.
La méthode renvoie une liste de chaines de caractères qui sont les différents morceaux.

```
chaine = 'les␣maths␣sont␣incompréhensibles'
t = chaine.split('␣')
# t contient ['les', 'maths', 'sont', 'incompréhensibles']
# les espaces sont enlevés
```

6.b. Transformer liste de chaines de caractères en une seule chaine de caractère

Ceci peut se faire avec une boucle et une succession de concaténation. Cependant, il existe une méthode déjà implémentée dans Python, la méthode join.
Sa syntaxe est la suivante : `'séparateur'.join(listedestring)`

```
liste=['hello','world','bonjour','le','monde']
c = '␣'.join(liste)
# c contient 'hello world bonjour le monde'
```

7. Test d'appartenance

Il est possible de tester si une sous-chaine se retrouve dans une chaine à l'aide du mot clé in avec la syntaxe `chaine1 in chaine2` qui renvoie un booléen.

```
bool  = 'e'  in 'bonjour' #bool  contiendra False
bool2 = 'j'  in 'bonjour' #bool2 contiendra True
bool3 = 'jo' in 'bonjour' #bool3 contiendra True
bool4 = 'oj' in 'bonjour' #bool4 contiendra False
```

Toutefois, il faudra apprendre à coder soi-même cette fonction pour évaluer son cout en terme de temps de calcul. Nous le feront dans le chapitre sur la complexité.

Table des matières