chapitre1_prof

August 26, 2026

1 Chapitre 1 : Parcours des listes (cb4c-6948113)

1.1 I- Le vocabulaire

- Une liste est **itérable** : on peut parcourir successivement ses éléments.
- Une liste est **indexable** (ou indiçable) : on peut accéder à chaque élément grâce à son indice.

1.2 II- Présentation des listes (code quizinière: AZE7KQ)

1.2.1 a) initialisation ou création d'une liste vide

[1]:
```
# Initialisation d'une liste vide
L=[]
```

1.2.2 b)Création d'une liste par extension (avec des éléments souhaités)

Une liste L se présente ainsi :

$$L = [elt_1, elt_2, elt_3, .., elt_n]$$

Cette liste contient n éléments et on obtient cette longueur de liste avec l'instruction : len(L)

Liste	elt_1	elt_2	...	elt_{n-1}	elt_n
Indice positif	0	1	...	$n-2$	$n-1$
Indice négatif	$-\text{len}(L)$	$-\text{len}(L)+1$	...	-2	-1

1.2.3 c)Obtenir un élément d'une liste

[3]:
```
# Prenon l'exemple de la liste suivante :
L=[5,14,9,15]
print(L[0])#renvoie la valeur du 1e élément (ici 5)
print(L[1])#renvoie la valeur du 2e élément (ici 14)
print(L[-1])#renvoie le dernier élément tout comme L[len(L)-1]
print(L[len(L)-1])# renvoie le dernier élément
# x in L renvoie True si x est dans L (False sinon)
print(14 in L)
```

```
5
14
15
15
True
```

1.2.4 d)Obtenir une sous liste L1 (une tranche) à partir de L

L1=L[d:f:p] est une sous liste de L avec : - d indice de début - f indice de fin exclu - p est le pas c'est-à-dire l'intervalle entre deux valeurs

Quelques raccourcis à connaître : - Par défaut p=1 si on écrit L[d:f] ce qui revient à sélectionner toutes les valeurs de l'indice d à f exclu - L[d:] revient à sélectionner les valeurs de l'indice d jusqu'à la fin incluse - L[:f] revient à sélectionner toutes les valeurs de l'indice 0 inclus à l'indice f exclu

[28]:
```
L=[5,12,14,18]
#  0, 1, 2, 3
print(L[1:4])#[12,14,18]
print(L[1:])#[12,14,18]
print(L[1:len(L)])#[12,14,18]
print(L[:-1])#[5,12,14] #tous les éléments sauf le dernier
print(L[::2])#[5,14] ici le pas est de 2
```

```
[12, 14, 18]
[12, 14, 18]
[12, 14, 18]
[5, 12, 14]
[5, 14]
```

1.2.5 e)Modifier un ou des éléments d'une liste

- L[i]=a affecte la valeur a à l'indice i dans L
- L[i:j+1]=L2 affecte la liste L2 dans L entre l'indice i et j inclus

[32]:
```
L=[5,12,14,18]
#  0, 1, 2, 3=len(L)-1
# -4,-3,-2,-1
L[0]=10#on modifie la 1e valeur de L
print(L)
L[-1]=20#L[len(L)-1]=20
print(L)
L2=[11,11,11]
L[1:4]=L2# remplace la tranche à partir de l'indice 1 par les éléments de L2
print(L)
```

```
[10, 12, 14, 18]
[10, 12, 14, 20]
[10, 11, 11, 11]
```

[8]:
```
#inversion de la liste
L=[0,1,2,3,4,5]
print(L[::-1])
print(L[len(L)-1::-1])#L[len(L)-1:-1:-1] renvoie [] car indice de départ et␣
 ↪arrivée confondus
print(L[-1:-len(L)-1:-1])
```

```
[5, 4, 3, 2, 1, 0]
[5, 4, 3, 2, 1, 0]
[5, 4, 3, 2, 1, 0]
```

[6]:
```
#rq : méthode insert
L=[0,1,2,4,5]
L.insert(3,3.5)#insère la valeur 3.5 à l'indice 3
print(L)
#fonction del
del(L[3])#supprime l'élément d'indice 3
print(L)
```

```
[0, 1, 2, 3.5, 4, 5]
[0, 1, 2, 4, 5]
```

1.2.6 f) Quelques opérations sur les listes

- la méthode append : L.append(x) rajoute x à la fin de L
- Concaténation de deux listes L1 et L2 : L1+L2
- L1*3 revient à L1+L1+L1

[7]:
```
L=[0,1,2,3,4]
L.append(5)
L2=[6,7]
L3=L+L2
print(L3*3)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7]
```

[1]:
```
#on peut évoquer la méthode extend qui permet de concaténer en place
L1=[0,1]
L2=[2,3]
print(id(L1))
L1.extend(L2)
print(L1,id(L1))
#au passage ajouter une liste avec append n'enlève pas les []:
L1=[0,1]
L2=[2,3]
L1.append(L2)
print(L1)
```

```
2995907823360
[0, 1, 2, 3] 2995907823360
```

```
[0, 1, [2, 3]]
```

1.3 III- Parcours des listes à l'aide de boucles (code quizinière : MZQ4Q8)

On distingue deux types de boucles :

1.3.1 a) La boucle non conditionnelle "for":

- Itération sur les valeurs d'une liste : La boucle *for* permet de récupérer à chaque itération un élément d'une liste L (le nombre d'itérations est borné car la liste L est finie).

[11]:
```
#Exemple 1:
L=[0,1,2,3,4]
for i in L:#itération sur chaque élément de L : i sera affectée successivement à␣
 ↪chaque valeur de L
    print(i)
```

```
0
1
2
3
4
```

[12]:
```
#Exemple 2 :
somme=0#initialisation
L=[0,1,2,3,4,5]
for i in L:
    somme=somme+i# on obtient la somme des éléments de L
print(somme)
```

```
15
```

- Itération sur les indices d'une liste : on utilise alors `range(d, f, p)` qui permet de parcourir, avec une boucle `for`, des entiers allant de `d` inclus à `f` exclu, avec un pas `p`.On retiendra que `range(f)` permet de parcourir tous les entiers allant de `0` inclus à `f` exclu, avec un pas de 1. Cela permet notamment de parcourir tous les indices d'une liste `L` avec `range(len(L))`.

[13]:
```
#Exemple 1
for i in range(0,5,2):
    print(i)
```

```
0
2
4
```

[14]:
```
#Exemple 2:
for i in range(5):
    print(i)
```

```
0
1
```

```
2
3
4
```

[15]:
```
#exemple 3:
L=[12,14,15,16]
n=len(L)
for i in range(n):
    print(i)
```

```
0
1
2
3
```

[17]:
```
#exemple 4:
L=[12,14,15,16]
n=len(L)
for i in range(n):
    print(L[i])# on peut aussi récupérer les valeurs de L
```

```
12
14
15
16
```

[24]:
```
#Exemple 5:
L=[10,14,15,16]
#i= 0, 1, 2, 3 => dernier indice =len(L)-1 avec len(L)=4
somme=0
n=len(L)
for i in range(n):# i=0,1,2,3 !!!!!!!! ici i c'est l'indice et i != L[i]
    somme=somme+L[i]
print(somme)
```

```
55
```

[33]:
```
#Compléments
#RQ : si range ne trouve pas de valeur car mauvais paramétrage => pas d'itération
for i in range(-5,0,-1):
    print(i)#aucune itération
for i in range(-5,0,1):
    print(i)
#la syntaxe est presque la même qu'avec le slice de liste sauf que le séparateur␣
 ↪est , (et pas :)
#A noter range(len(L)-1,-1,-1) => conduit bien à itérer correctement (au␣
 ↪contraire des listes)
L=[0,1,2,3]
for i in range(len(L)-1,-1,-1):
```

```
    print(i)
print(L[len(L)-1:-1:-1])#ne pas mélanger les indices lus de gauche à droite avec␣
 ↪ceux lus de droite à gauche
```

```
-5
-4
-3
-2
-1
3
2
1
0
[]
```

1.3.2 b) Les boucles conditionnelles "while" :

La boucle `while` répète une séquence d'instructions tant qu'une condition est vraie. Il faut donc généralement modifier, dans la boucle, une variable intervenant dans la condition afin que celle-ci devienne fausse et que la boucle s'arrête.

[26]:
```
#exemple 1:
i=0
while i<5:
    print("Bonjour")
    i=i+1#essentiel pour éviter de boucler à l'infini
```

```
Bonjour
Bonjour
Bonjour
Bonjour
Bonjour
```

Cette boucle va aussi nous permettre de parcourir des listes:

[30]:
```
somme=0#initialisation
L=[0,1,2,3,4,5]
i=0#initialisation
while i<len(L):
    somme=somme+L[i]
    i=i+1#essentiel pour parcourir la liste et stopper la boucle
print(somme)
```

```
15
```

Rq : ici i joue à la fois le rôle d'indice et de compteur initialisé à 0, qui permet par exemple de connaître le nombre de passages dans la boucle while

Compléments :

On peut donc générer des listes :

- Par extension (manuellement) L=[0,1,2,3,4]
- Par conversion L=list(range(5))
- par compréhension : liste=[i**2 for i in range(5) if i%2==0]
- Par ajout successif avec une boucle for et la méthode append

[36]:
```
# Par extension
L1 = [0, 1, 2, 3, 4]

# Par conversion
L2 = list(range(5))

# Par ajouts successifs
L3 = []
for i in range(5):
    L3.append(i)

# Par compréhension
L4 = [i**2 for i in range(5) if i % 2 == 0]

print(L1)
print(L2)
print(L3)
print(L4)
```

```
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4]
[0, 4, 16]
```

[10]:
```
#Enfin signaler qu'il est possible de forcer l'arrêt d'une boucle
#(typiquement dans le cas d'une boucle while) => à éviter cependant
i=0
while i<100:
    if i==10:
        break
    i=i+1
print(i)
```

```
10
```

[2]:
```
#exemple 1 : obtenie les carrés d'une liste

def carre1(L):
    return [i**2 for i in L]
def carre2(L):
    L_new=[]
    for i in L:
        L_new.append(i**2)
```

```
    return L_new
def carre3(L):# en place !
    i=0
    while i<len(L):
        L[i]=L[i]**2
        i=i+1
    return L
def carre4(L):# en place !
    for i in range(len(L)):
        L[i]=L[i]**2
    return L

L=[0,1,2,3,4,5,6,7,8,9]
print(carre1(L))
print(carre2(L))
print(carre3(L))
print(carre4(L))
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 16, 81, 256, 625, 1296, 2401, 4096, 6561]
```

[ ]:
```
#exemple 2 : exponentiation naïve
def expo1(x,n):
    if n<0:
        raise ValueError("n ne doit pas être négatif")
    else :
        r=1
        for i in range(1,n+1):
            r=r*x
    return r

def expo2(x,n):
    if n<0:
        raise ValueError("n ne doit pas être négatif")
    else :
        r=1
        i=1
        while i<=n:
            r=r*x
            i=i+1
    return r

print(expo1(10,3))
print(expo2
```

[4]:
```
#exemple 3 : trouver le min et le max d'une liste:
def min_max(L):
    mini,maxi=L[0],L[0]
    i_min,imax=0,0
    for i in range(1,len(L)):
        if maxi<L[i]:
            maxi,i_max=L[i],i
        elif mini>L[i]:
            mini,i_min=L[i],i
    return maxi,i_max,mini,i_min
min_max([0,1,2,3,4,5,6,7,8,9])
```

[4]:
```
(9, 9, 0, 0)
```