Les Listes

October 22, 2023

1 Les Listes - Généralités

[1]:
```
# Exemple des listes
L = [12, 45, 3.667, True, [12,6,45], "abc"]
M = [12, 45, 45, 78]
```

[2]:
```
# Déclarer une liste vide
L = []

#ou
L = list()
```

[3]:
```
# La fonction len retoune le nombre des éléments d'une liste
L = [12, 45, 3.667, True, [12,6,45], "abc"]
print(len(L))
```

```
6
```

[4]:
```
# Accéder aux éléments de la liste à l'aide des indices positifs
L = [12, 45, 3.667, True, [12,6,45], "abc"]
print("L =",L)
print("L[0] =", L[0])
print("L[2] =", L[2])
print("L[5] =", L[5])
```

```
L = [12, 45, 3.667, True, [12, 6, 45], 'abc']
L[0] = 12
L[2] = 3.667
L[5] = abc
```

[5]:
```
# Accéder aux éléments de la liste à l'aide des indices négatifs
L = [12, 45, 3.667, True, [12,6,45], "abc"]
print("L =",L)
print("L[-1] =", L[-1])
print("L[-3] =", L[-3])
print("L[-2] =", L[-2])
print("L[-6] =", L[-6])
```

```
L = [12, 45, 3.667, True, [12, 6, 45], 'abc']
L[-1] = abc
L[-3] = True
L[-2] = [12, 6, 45]
L[-6] = 12
```

[6]:
```
# Extraction des sous listes (Slicing)
M = [12, 45, 3.667, True, [12,6,45], "abc", 4, 88, 12.7654, 13]
print("M =", M)
print("M[3: 9] =", M[3: 9])
print("M[4: 6] =", M[4: 6])
print("M[-8: -3] =", M[-8: -3])
print("M[: 6] =", M[: 6])
print("M[6:] =", M[6:])
print("M[6: 2] =", M[6: 2])
```

```
M = [12, 45, 3.667, True, [12, 6, 45], 'abc', 4, 88, 12.7654, 13]
M[3: 9] = [True, [12, 6, 45], 'abc', 4, 88, 12.7654]
M[4: 6] = [[12, 6, 45], 'abc']
M[-8: -3] = [3.667, True, [12, 6, 45], 'abc', 4]
M[: 6] = [12, 45, 3.667, True, [12, 6, 45], 'abc']
M[6:] = [4, 88, 12.7654, 13]
M[6: 2] = []
```

[7]:
```
# Extraction des sous listes (Slicing) - Utilisation du pas
M = [12, 45, 3.667, True, [12,6,45], "abc", 4, 88, 12.7654, 13]
print("M =", M)
print("M[3: 9: 2] =", M[3: 9: 2])
print("M[:: 2] =", M[:: 2])
print("M[:: 3] =", M[:: 3])
print("M[6: 2: -1] =", M[6: 2: -1])
print("M[:: -1] =", M[:: -1])
```

```
M = [12, 45, 3.667, True, [12, 6, 45], 'abc', 4, 88, 12.7654, 13]
M[3: 9: 2] = [True, 'abc', 88]
M[:: 2] = [12, 3.667, [12, 6, 45], 4, 12.7654]
M[:: 3] = [12, True, 4, 13]
M[6: 2: -1] = [4, 'abc', [12, 6, 45], True]
M[:: -1] = [13, 12.7654, 88, 4, 'abc', [12, 6, 45], True, 3.667, 45, 12]
```

[8]:
```
# Modifications des éléments d'une liste à l'aide des indices
P = [12, 44, 66, 88, 13]
P[1] = 100
P[-1] = 200
print(P)
```

```
[12, 100, 66, 88, 200]
```

[9]:
```
# Modifications des éléments d'une liste à l'aide des indices
P = [12, 44, 66, 88, 13, 16, 52]
P[3] = 1000
P[-2] = 2000
P[0] = 3000
print(P)
```

```
[3000, 44, 66, 1000, 13, 2000, 52]
```

[10]:
```
# La concaténation des listes
K = [12, 44, 3]
M = [1, 5, 7]
S = K + M
print("K =",K)
print("M =",M)
print("S =",S)
```

```
K = [12, 44, 3]
M = [1, 5, 7]
S = [12, 44, 3, 1, 5, 7]
```

[11]:
```
# La concaténation des listes
K = [12, 44, 3]
M = [1, 5, 7]
S = M + K
print("K =",K)
print("M =",M)
print("S =",S)
```

```
K = [12, 44, 3]
M = [1, 5, 7]
S = [1, 5, 7, 12, 44, 3]
```

[12]:
```
# La multiplication d'une liste par un entier positif (int)
K = [12, 44, 3]
M = 3 * K
print("K =",K)
print("M =",M)
```

```
K = [12, 44, 3]
M = [12, 44, 3, 12, 44, 3, 12, 44, 3]
```

[13]:
```
# La multiplication d'une liste par un entier positif (int)
K = [0] * 10
print("K =",K)
```

```
K = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

[14]:
```
# Le test d'appartenance d'un élément dans une liste
K = [12, 44, 3]
A = 12 in K
print(A)
```

```
True
```

[15]:
```
# Le test d'appartenance d'un élément dans une liste
K = [12, 44, 3]
A = 120 in K
print(A)
```

```
False
```

[16]:
```
# Le test d'appartenance d'un élément dans une liste
K = [12, 44, 3]

if 44 in K:
  print("44 appartient à K")
else:
  print("44 n'appartient pas à K")
```

```
44 appartient à K
```

[17]:
```
# Le test d'appartenance d'un élément dans une liste
K = [12, 44, 3]

if 440 in K:
  print("440 appartient à K")
else:
  print("440 n'appartient pas à K")
```

```
440 n'appartient pas à K
```

[18]:
```
# Le parcours des éléments d'une liste à l'aide de la boucle for
L = [12, 45, 3.667, True, [12,6,45], "abc"]
for x in L:
  print(x)
```

```
12
45
3.667
True
[12, 6, 45]
abc
```

[19]:
```
# Le parcours des indices d'une liste à l'aide de la boucle for
L = [12, 45, 3.667, True, [12,6,45], "abc"]
```

```
for x in range(len(L)):
  print(x)
```

```
0
1
2
3
4
5
```

[20]:
```
# Le parcours des indices d'une liste à l'aide de la boucle for
L = [12, 45, 3.667, True, [12,6,45], "abc"]
for x in range(len(L)):
  print(L[x])
```

```
12
45
3.667
True
[12, 6, 45]
abc
```

2 Les Listes - Fonctions Prédéfinies

- La fonction max(L) : retourne le maximum de la liste L (La liste L ne doit contenir que des nombres ou que des chaines de caractères) ;
- La fonction min(L) : retourne le minimum de la liste L (La liste L ne doit contenir que des nombres ou que des chaines de caractères) ;
- La fonction sum(L) : retourne la somme des éléments de la liste L (La liste L ne doit contenir que des nombres) ;
- La fonction sorted(L) : retourne une copie de la liste L, triée dans l'ordre croissant (La liste L ne doit contenir que des nombres) ;
- La fonction sorted(L, reverse = True) : retourne une copie de la liste L, triée dans l'ordre décroissant (La liste L ne doit contenir que des nombres).

[21]:
```
# La fonction max
L = [12, 44, 67, 54, 12.3, 11.98, 5]
m = max(L)
print(m)
```

```
67
```

[22]:
```
# La fonction min
L = [12, 44, 67, 54, 12.3, 11.98]
m = min(L)
print(m)
```

```
11.98
```

[23]:
```
# La fonction sum
L = [1, 4, 6, 5, 12.3, 2]
s = sum(L)
print(s)
```

```
30.3
```

[24]:
```
# la fonction sorted
L = [12, 44, 67, 54, 12.3, 11.98, 5]
M = sorted(L)
print("L =", L)
print("M =", M)
```

```
L = [12, 44, 67, 54, 12.3, 11.98, 5]
M = [5, 11.98, 12, 12.3, 44, 54, 67]
```

[25]:
```
# la fonction sorted
L = [12, 44, 67, 54, 12.3, 11.98, 5]
M = sorted(L, reverse = True)
print("L =", L)
print("M =", M)
```

```
L = [12, 44, 67, 54, 12.3, 11.98, 5]
M = [67, 54, 44, 12.3, 12, 11.98, 5]
```

3 Les Listes - Méthodes Prédéfinies

- L.append(x) : Ajoute l'objet x à la fin de la liste L ;
- L.insert(i, x) : Insère (ajoute) l'objet x dans la position (indice) i dans la liste L ;
- L.extend(M) : Ajoute les éléments de la liste M à la fin de la liste L ;
- L.remove(x) : Supprime la première occurence de x dans L ;
- L.pop(i) : Supprime l'élément L[i] et elle le retoune ;
- L.reverse() : Inverse l'ordre des éléments de la liste ;
- L.sort() : Trie la liste dans l'ordre croissant (La liste L ne doit contenir que des nombres ou que des chaines de caractères) ;
- L.sort(reverse = True) : Trie la liste dans l'ordre décroissant (La liste L ne doit contenir que des nombres ou que des chaines de caractères) ;
- L.clear() : Vider la liste ;
- L.index(x) : Retourne l'indice de la première occurence de x dans L ;
- L.count(x) : Retourne le nombre d'occurence de x dans L.

[26]:
```
# La méthode .append()
L = [12, 5, 88, 4]
L.append(100)
print(L)
```

```
[12, 5, 88, 4, 100]
```

[27]:
```
# La méthode .append()
L = [12, 5, 88, 4]
L.append(100)
L.append(50)
print(L)
```

```
[12, 5, 88, 4, 100, 50]
```

[28]:
```
# La méthode .insert()
L = [12, 5, 88, 4]
L.insert(2, 100)
print(L)
```

```
[12, 5, 100, 88, 4]
```

[29]:
```
# La méthode .extend()
L = [12, 5, 88, 4]
M = [1, 2, 3]
L.extend(M)
print("L =",L)
print("M =",M)
```

```
L = [12, 5, 88, 4, 1, 2, 3]
M = [1, 2, 3]
```

[30]:
```
# La méthode .extend()
L = [12, 5, 88, 4]
M = [1, 2, 3]
M.extend(L)
print("L =",L)
print("M =",M)
```

```
L = [12, 5, 88, 4]
M = [1, 2, 3, 12, 5, 88, 4]
```

[31]:
```
# La méthode .remove()
L = [12, 5, 88, 4]
L.remove(88)
print(L)
```

```
[12, 5, 4]
```

[32]:
```
# La méthode .remove()
L = [12, 5, 88, 4, 88, 54, 88]
L.remove(88)
print(L)
```

```
[12, 5, 4, 88, 54, 88]
```

[33]:
```
# La méthode .pop()
L = [12, 5, 88, 4, 88, 54, 88]
s = L.pop(1)
print("L =",L)
print("s =",s)
```

```
L = [12, 88, 4, 88, 54, 88]
s = 5
```

[34]:
```
# La méthode .pop()
L = [12, 5, 88, 4, 88, 54, 88]
s = L.pop()
print("L =",L)
print("s =",s)
```

```
L = [12, 5, 88, 4, 88, 54]
s = 88
```

[35]:
```
# La méthode .reverse()
L = [12, 5, 88, 4]
L.reverse()
print(L)
```

```
[4, 88, 5, 12]
```

[36]:
```
# La méthode .sort()
L = [12, 5, 88, 4]
L.sort()
print(L)
```

```
[4, 5, 12, 88]
```

[37]:
```
# La méthode .sort()
L = [12, 5, 88, 4]
L.sort(reverse = True)
print(L)
```

```
[88, 12, 5, 4]
```

[38]:
```
# La méthode .clear()
L = [12, 5, 88, 4]
L.clear()
print(L)
```

```
[]
```

[39]:
```
# La méthode .index()
L = [12, 55, 7, 34, 87]
```

```
i = L.index(7)
print(i)
```

```
2
```

[40]:
```
# La méthode .index()
L = [12, 55, 7, 34, 87, 45, 34]
i = L.index(34)
print(i)
```

```
3
```

[41]:
```
# La méthode .count()
L = [12, 55, 7, 34, 87, 45, 34]
c = L.count(34)
print(c)
```

```
2
```

[42]:
```
# La méthode .count()
L = [12, 55, 7, 34, 87, 45, 34]
c = L.count(12)
print(c)
```

```
1
```

[43]:
```
# La méthode .count()
L = [12, 55, 7, 34, 87, 45, 34]
c = L.count(88)
print(c)
```

```
0
```

4 Exercices

4.1 Exercice 1

Ecrire une fonction affiche_inverse(L) qui prend en paramètre une liste L et qui affiche ses éléments en inverse (On commence par le dernier élément et on finit par le premier élément de la liste).

[44]:
```
# Méthode 1
def affiche_inverse(L):
  for k in range(len(L) - 1, -1, -1):
    print(L[k])
```

[45]:
```
# Méthode 2
def affiche_inverse(L):
  for k in range(-1, -len(L) - 1, -1):
```

```
    print(L[k])
```

4.2 Exercice 2

1. Ecrire une fonction max_liste(L) qui prend en paramètre une liste qui ne contient que des nombres (int ou float) et qui retoune le nombre maximum de la liste.
2. Ecrire une fonction min_liste(L) qui prend en paramètre une liste qui ne contient que des nombres (int ou float) et qui retoune le nombre minimum de la liste.
3. Ecrire une fonction sum_liste(L) qui prend en paramètre une liste qui ne contient que des nombres (int ou float) et qui retoune la somme des éléments de la liste.

[46]:
```
# 1 - Méthode 1
def max_liste(L):
  m = L[0]
  for i in range(len(L)):
    if L[i] > m:
      m = L[i]
  return m
```

[47]:
```
# 1 - Méthode 2
def max_liste(L):
  m = L[0]
  for x in L:
    if x > m:
      m = x
  return m
```

[48]:
```
# 2 - Méthode 1
def min_liste(L):
  m = L[0]
  for i in range(len(L)):
    if L[i] < m:
      m = L[i]
  return m
```

[49]:
```
# 2 - Méthode 2
def min_liste(L):
  m = L[0]
  for x in L:
    if x < m:
      m = x
  return m
```

[50]:
```
# 3 - Méthode 1
def sum_liste(L):
  s = 0
```

```
  for i in range(len(L)):
    s = s + L[i]
  return s
```

[51]:
```
# 3 - Méthode 2
def sum_liste(L):
  s = 0
  for x in L:
    s = s + x
  return s
```

4.3 Exercice 3

Ecrire une fonction indice(L, x) qui prend en paramètre une liste L et un objet x. La fonction retourne l'indice de la première occurence de x dans L si x existe dans L. Sinon la fonction retourne -1. Sans utiliser la méthode prédifinie .index().

[52]:
```
def indice(L, x):
  for i in range(len(L)):
    if L[i] == x:
      return i
  return -1
```

4.4 Exercice 4

Ecrire une fonction compte(L, x) qui prend en paramètre une liste L et un objet x. La fonction retourne le nombre d'occurence de x dans L.

[53]:
```
def compte(L, x):
  s = 0
  for y in L:
    if y == x:
      s = s + 1
  return s
```

4.5 Exercice 5

Un altimètre est un instrument de mesure permettant de déterminer la distance verticale entre un point et une hauteur de référence. Lors d'une randonnée en montagne, Alice étalonne son altimètre à son point de départ, puis mesure à chaque heure l'altitude relative atteinte. À la fin de sa randonnée, elle obtient une liste d'entiers naturels qu'elle range dans une liste Python nommée alt.

La randonnée d'Alice a duré 10 heures, elle a atteint une altitude relative de 300m au bout d'une heure, de 500m au bout de deux heures, etc.

Alice souhaite maintenant calculer certaines valeurs relatives à son parcours. 1. Quelle fonction déjà existante en Python lui permet de calculer la durée de sa randonnée ? 2. Définir en Python une fonction nommée altmax, qui prend en argument une liste alt et qui retourne l'altitude relative maximale atteinte lors de sa randonnée. Par exemple, dans le cas de l'illustration numérique donnée ci-dessus la fonction devra renvoyer la valeur 1000. 3. Alice cherche maintenant à savoir à quel moment son ascension a été la plus rapide en calculant le dénivelé maximal réalisé en une heure. Elle cherche donc la plus grande différence entre deux emplacements consécutifs de la liste. Par exemple, pour la liste donnée en illustration le dénivelé maximal est égal à 400 et a été réalisé entre la troisième et la quatrième heure. Définir en Python une fonction nommée denivmax prenant en argument une liste alt et retournant le dénivelé maximal réalisé en une heure durant sa randonnée. 4. Modifier la fonction précédente pour qu'elle retourne non pas le dénivelé maximal mais l'heure à laquelle débute la réalisation de ce dénivelé. Pour la liste donnée en exemple, cette fonction devra donc retourner la valeur 3. 5. Définir une fonction nommée denivtotal retournant la somme des dénivelés positifs réalisés durant cette randonnée. Pour l'exemple donné en illustration cette fonction devra donc retourner la valeur 1200. 6. Enfin, on appelle sommet toute altitude relative strictement supérieure à l'altitude qu'elle précède et à l'altitude qui lui succède dans la liste alt. Dans l'exemple qui nous sert à illustrer cet exercice, la randonnée d'Alice présente trois sommets de valeurs 1000, 900 et 600 atteints à la quatrième, sixième et huitième heure de marche. Rédiger fonction sommets retournant la liste des sommets de la randonnée.

[54]:
```
alt = [0, 300, 500, 600, 1000, 800, 900, 500, 600, 200, 0]
```

[55]:
```
# 1
# Il s'agit de la fonction len, en effet la durée de la randonnée est égale à␣
 ↪len(alt) - 1
duree = len(alt) - 1
print(duree)
```

```
10
```

[56]:
```
# 2
def altmax(alt):
  m = alt[0]
  for x in alt:
    if x > m:
      m = x
  return m
```

[57]:
```
# 3
def denivmax(alt):
  m = alt[1] - alt[0]
  for i in range(1, len(alt) - 1):
    if alt[i + 1] - alt[i] > m:
      m = alt[i + 1] - alt[i]
  return m
```

[58]:
```
# 4
def denivmax(alt):
  m = alt[1] - alt[0]
  k = 0
  for i in range(1, len(alt) - 1):
    if alt[i + 1] - alt[i] > m:
      m = alt[i + 1] - alt[i]
      k = i
  return k
```

[59]:
```
# 5
def denivtotal(alt):
  s = 0
  for i in range(len(alt) - 1):
    if alt[i + 1] - alt[i] > 0:
      s = s + (alt[i + 1] - alt[i])
  return s
```

[60]:
```
# 6
def sommets(alt):
  L = []
  for i in range(1, len(alt) - 1):
    if alt[i + 1] < alt[i] and alt[i] > alt[i - 1]:
      L.append(alt[i])
  return L
```