BTS
BREVET DE TECHNICIEN SUPÉRIEUR
LYCÉE QUALIFIANT MOHAMED V
ESSAOUIRA

PROGRAMMATION EN PYTHON

LISTE & TUPLE

Fatiha BENDAIDA

DWFS 1

2024/2025

LISTES

- Les variables, telles que nous les avons vues, ne permettent de stocker qu'une seule donnée à la fois. Or, pour de **nombreuses données**, comme cela est souvent le cas, des variables distinctes seraient beaucoup trop lourdes à gérer.

- Heureusement, Python **proposent des structures** de données **permettant de stocker** l'ensemble de ces données dans **une variable commune**. Ainsi, pour accéder à ces valeurs il suffit de parcourir la variable en utilisant les indices.

LISTES

Définition

Une liste est une structure de données qui permet de stocker plusieurs valeurs mais de type hétérogène **simple** où **composés** .

Création : On peut construire une liste de plusieurs manières :

- Par la **donnée explicite** des éléments, entre crochets, séparés par des virgules :

```
>>> L1=[]  # liste vide (ou bien L1=list() )
>>> L2=[1, 15, 20, 35]  # liste des entiers
>>> L3=[True, 12, ''hello'', 12.2]
```

LISTES

- Par **concaténation** de listes (à l'aide de +) :

```
>>> L=[10, 12, 33] + [14, 52, 42, 56]
>>> L
[10, 12, 33, 14, 52, 42, 56]
>>> L=[-5] + L
>>> L
[-5, 10, 12, 33, 14, 52, 42, 56]
```

- ***list(iterable)*** permet de fabriquer une liste a partir d'un itérable :

```
>>> L=list("hello")
>>> print(L)
['h', 'e', 'l', 'l', 'o']
```

```
>>> L=list(range(5))
>>> print(L)
[0,1,2,3,4]
```

LISTES

- **Par compréhension**, très pratique avec le syntaxe suivant :

L=[f(x) for x in iterable if P(x)],

où f(x) est une expression dépendant (ou non) de x, et P(x) est une expression booléenne (facultative) :

```
>>> L=[2*i for i in range(10) ]
>>> print(L)
[0, 2,4, 6, 8, 10, 12, 14, 16, 18]
```

```
>>> L=[i*i for i in range(10) if i%2==0]
>>> print(L)
[0, 4, 16, 36, 64]
```

LISTES

- **Exercice 1:**

Ecrire une fonction ***liste_alea(n)*** qui reçoit en paramètre un entier n et qui de retourne une liste contenant n entiers générer aléatoirement dans l'intervalle [0,20].

```
>>> liste_alea(5)
[1, 3, 5, 15,0]
```

```
13  from random import*
14  def  liste_alea(n):
15      L=[]
16      for i in range(n):
17          L=[randint(0,20)]+L
18      return L
```

```
13  from random import*
14  def   liste_alea(n):
15      return [randint(0,20) for i in range(n)]
16
```

LISTES

- **Exercice 2:**

Ecrire une fonction ***diviseur(n)*** qui reçoit en paramètre un entier n et qui de retourne une liste de ses diviseurs.

```
>>> diviseur (15)
[1, 3, 5, 15]
```

```
1  def  diviseur(n):
2       L=[]
3       for i in range(1,n+1):
4           if n%i==0:
5               L=L+[i]
6       return L
```

```
1  def  diviseur(n):
2       return [i for i in range(1,n+1) if n%i==0]
```

LISTES

Accès aux valeurs d'une liste :

On peut accéder aux valeurs de la liste en indiquant l'indice ou les indices entre crochets qui peuvent être positifs ou bien négatifs :

Exemple :

Liste des matières que vous étudiez :

m=["MATH", "PHY", "SI", "INFO", "FR", "EN", "TRA", "SPO"]

MATH	PHY	SI	INFO	FR	EN	TRA	SPO
0	1	2	3	4	5	6	7
-8	-7	-6	-5	-4	-3	-2	-1

LISTES

- **Exercice 3:**

Ecrire une fonction ***liste_alea_trie(n)*** qui reçoit en paramètre un entier n et qui de retourne une liste contenant n entiers générer aléatoirement triées dans l'ordre croissant.

```
>>> liste_alea_trie(5)
[1, 2, 5, 10,10]
```

```
1  from random import randint
2  def   liste_alea_trie(n):
3      L=[randint(0,20)]
4      for  i  in range(n-1):
5          a= randint(0,20)
6          while(a<L[-1]):
7              a= randint(0,20)
8          L=L+[a]
9      return L
10
11 print(liste_alea_trie(5))
```

LISTES

Slicing (tranchage) :

On peut créer une nouvelle liste en extrayant certains éléments d'une liste. Pour extraire les éléments d'indice entre **d** inclus et $f \geq d$ exclus avec un pas **p**, on utilise

L[d :f :p]

la liste obtenue est donc composée des éléments

L[d], L[d+p],... L[d+kp] tel que $d + kp < f$

```
>>> L=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> L[1: 7: 3]
[1, 4]
>>> L[1: 7: -1]
[]
>>> L[7: 1: -1]
[7, 6, 5, 4, 3, 2]
```

LISTES

Slicing (tranchage) :

```
>>> L=[2, 5, 6, 8, 33, 55, 1, -5, -23]
```

```
>>> L[3:5]
```

```
[8, 33]
```

```
>>> L[:5]
```

```
[2, 5, 6, 8, 33]
```

```
>>> L[4:]
```

```
[33, 55, 1, -5, -23]
```

```
>>> L[:]
```

```
[2, 5, 6, 8, 33, 55, 1, -5, -23]
```

```
>>> L[ : :1 ]
```

```
[2, 5, 6, 8, 33, 55, 1, -5, -23]
```

```
>>> L[ : : -1 ]
```

```
[-23, -5, 1, 55, 33, 8, 6, 5, 2]
```

LISTES

Opérations et fonctions sur les listes :

Soit L1 et L2 deux listes tels que :

```
>>> L1=[0, 1, 2, 3, 4]
>>> L2=[ 5, 6, 7, 8, 9]
```

- **Concaténation de deux listes (+)**

```
>>> L1+L2
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- **Répétition d'une liste (*)**

```
>>> L1*2          # (même chose avec  2*L1)
[0, 1, 2, 3, 4, 0, 1, 2, 3, 4]
```

- **Taille de la liste :**

```
>>> len(L1)
5
```

LISTES

Opérations et fonctions sur les listes :

Soit L1 et L2 deux listes tels que :

```
>>> L1=[0, 1, 2, 3, 4]
>>> L2=[ 5, 6, 7, 8, 9]
```

- **Concaténation de deux listes (+)**

```
>>> L1+L2
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- **Répétition d'une liste (*)**

```
>>> L1*2          # (même chose avec  2*L1)
[0, 1, 2, 3, 4, 0, 1, 2, 3, 4]
```

- **Taille de la liste :**

```
>>> len(L1)
5
```

LISTES

Opérations et fonctions sur les listes :

- **Test d'appartenance**

```
>>> 3 in [0,1,2,3,4]
True
```

- **Test de non appartenance**

```
>>> 5 not in [ 5, 6, 7, 8, 9]
False
```

- **Itéré les éléments de la liste**

```
>>> for x in [0,1,2,3,4] : print(x, end=" | ")
0|1|2|3|4
```

```
>>> for i in range(len(L1)) :
        print(L[i], end=":")
0:1:2:3:4
```

LISTES

Exercice 4: Ecrire une fonction **produit_scalaire(L1,L2)** qui retourne le produit scalaire ($\sum_{i=1}^{n} L1[i] * L2[i]$) entre les deux liste L1 et L2. (L1 et L2 peuvent être de taille différentes)

```
>>> L1=[ 1, 4, 6]
>>> L2=[-3, 5, 1]
>>> produit_scalaire(L1,L2)
23
```

```
1  def  produit_scalaire(L1,L2):
2      if  len(L1) != len(L2):
3          return False
4      else :
5          s=0
6          for i in range(len(L1)):
7              s+=L1[i]*L2[i]
8          return  s
```

LISTES

Opérations et fonctions sur les listes :

- **Enumerate**

Lorsque vous itérez sur une séquence, la **position** et la **valeur** correspondante peuvent être **récupérées en même temps** en utilisant la fonction ***enumerate()***

```
1  l1 = ["eat","sleep","repeat"]
2  obj1 = enumerate(l1)
3
4  print(list(obj1))
```

```
[(0, 'eat'), (1, 'sleep'), (2, 'repeat')]
```

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
```

LISTES

Opérations et fonctions sur les listes :

- **Maximum, minimum d'une liste**

```
>>> T=[ -1, 1, 6, 4, 7]

>>> max(T)
7
>>> min(T)
-1
```

- **Somme des élément d'une liste**

```
>>> sum(T)
17
```

```
>>> sum([])
0
```

LISTES

Méthodes sur les listes :

Python inclut la liste ci-dessous des méthodes de manipulation des listes :

méthode	description
`L.append(x)`	Ajoute `x` à la fin de `L`.
`L.extend(T)`	Ajoute les éléments de `T` à la fin de `L` (équivalent à `L+=T`).
`L.insert(i, x)`	Ajoute l'élément `x` en position `i` de `L`, en décalant les suivants vers la droite.
`L.remove(x)`	Supprime de la liste la première occurrence de `x` si `x` est présent, sinon produit une erreur.
`L.pop()`	Supprime le dernier élément de `L`, et le renvoie.
`L.pop(i)`	Supprime l'élément d'indice `i` de `L`, en décalant les suivants vers la gauche. Cette méthode renvoie l'élément supprimé.
`L.index(x)`	Retourne l'indice de la première occurrence de `x` dans `L` si `x` est présent, produit une erreur sinon.
`L.count(x)`	Retourne le nombre d'occurrences de `x` dans `L`.
`L.sort()`	Trie la liste `L` dans l'ordre croissant (en place).
`L.reverse()`	Renverse la liste (en place)

LISTES

```
>>> L=[2, 5, 6, 8, 33, 55, 1, -5, -23]
```

```
>>> L.insert(3,5)
```

```
[2, 5, 6, 5, 8, 33, 55, 1, -5, -23]
```

```
>>> L.pop()
```

```
-23
[2, 5, 6, 5, 8, 33, 55, 1, -5]
```

```
>>> L.index(5)
```

```
1
```

```
>>> L.count(5)
```

```
2
```

```
>>> L.append(-9)
```

```
[2, 5, 6, 5, 8, 33, 55, 1, -5, -9]
```

```
>>> L.sort()
```

```
[-9, -5, 1, 2, 5, 5, 6, 8, 33, 55]
```

```
>>>  L.extend([100,200])
```

```
[-9, -5, 1, 2, 5, 5, 6, 8, 33, 55,100,200]
```

LISTES

Exercice 6: (stupid sort)

1. Écrire une fonction **est_trie(t)**, qui prend en paramètre une liste t et qui renvoie **True** si la liste est triée et **False** sinon.

```
1  def est_trie(t):
2      test=[t[i]<=t[i+1] for i in range(len(t)-1)]
3      return False not in test
```

2. Un singe trie les cartes de la façon suivante : il prend les cartes, les jette en l'air, les ramasse puis regarde si elles sont triées. Si oui, il s'arrête, sinon il relance les cartes. Implémenter et tester ce tri.

N.B : la méthode **shuffle** du module **random** permet de réorganise l'ordre des éléments aléatoirement.

```
5  from random import shuffle
6  def stupid_sort(t):
7      while(not est_trie(t)):
8          shuffle(t)
9      return t
```

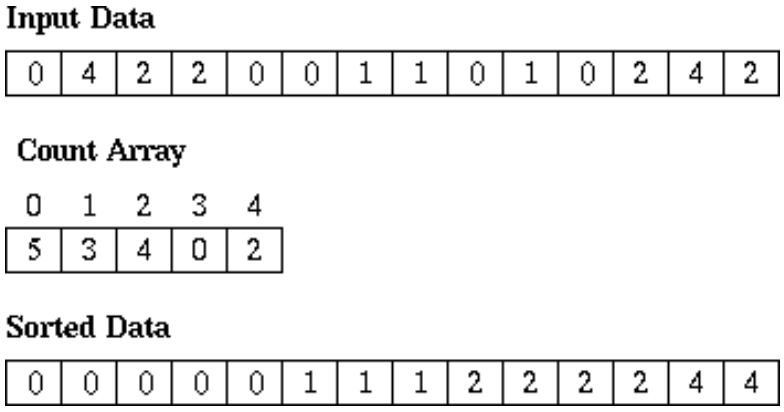
LISTES

Exercice 7: (Counting sort)

1. On suppose que tous les nombres sont compris entre 0 et M, où M est fixé. Afin de trier une liste t, le principe est le suivant :

- On fixe M = max(t) et on crée une liste **tiroirs** constitué de M + 1 zéros;
- on modifie la liste **tiroirs** de manière à ce que **tiroirs[k]** soit égal au nombre d'éléments de valeur k dans le tableau t
- à l'aide de la liste tiroirs, on trie la liste t en renvoyant une liste contenant dans l'ordre : tiroirs[0] × [0], tiroirs[1] × [1], tiroirs[2] × [2], …

Écrire une fonction **tri_par_denombrement(t)** qui implémente ce tri.




```python
def counting_sort(t):
    M=max(t)+1
    tiroirs=[0]*M
    for e in t:
        tiroirs[e]=t.count(e)
    L=[]
    for i,e in enumerate(tiroirs):
        L.extend([i]*e)
    return L

print(counting_sort([4,0,4,5,1,3,0]))
```

Les structures de contrôle `zip` et `map`

- **zip** : permet de parcourir plusieurs séquences en parallèle
- **map** : applique une méthode sur une ou plusieurs séquences

Remarque

map peut être beaucoup plus rapide que l'utilisation de **for**

Exemples

- **Utilisation de zip**

```
1  L1=[1,2,3]
2  L2=[10,0,9,6]
3  for x,y in zip(L1,L2):
4      print(x,"--",y)
```

Affichage

```
1 -- 10
2 -- 0
3 -- 9
```

LES STRUCTURES DE CONTRÔLE `ZIP` ET `MAP`

Exemples

- **Utilisation de map**

>>> S = '0123456789'

>>> print(list(map(int, S)))

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```
numbers = [1, 1, 2, 3, 4, 5]
resultat=list(map(lambda x: 2*x+1,numbers))
print(resultat)
```

```
l = ['sat', 'bat', 'cat', 'mat']

test = list(map(list, l))

print(test)
```

[['s', 'a', 't'], ['b', 'a', 't'], ['c', 'a', 't'], ['m', 'a ',' t ']]

LISTES

Exercice 8:

Un polynôme peut être représenté par une liste dont les éléments sont les coefficients du polynôme et les indices sont les exposants.

Exemple : le polynôme **$p(x)= 1-3x^2+4x^5$** sera représenté par la liste **p=[1, 0, -3, 0, 0, 4].**

1. Ecrire une fonction **generer_poly(n)** qui permet de générer aléatoirement un polynôme de degré n
2. Ecrire une fonction **liste_poly(L)** qui permet de retourner une liste des polynômes de degré respective les éléments de la liste L.

```
1  from random import randint
2  def generer_poly(n):
3      return [randint(-5,5) for i in range(n+1)]
```

```
5  def liste_poly(L):
6      result=list(map(generer_poly,L))
7      return result
```

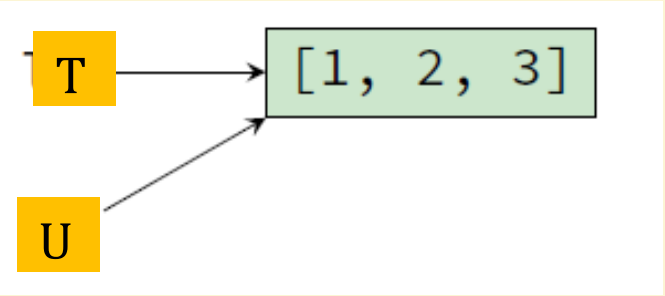
LISTES

Copie d'une liste simple:

Analysant le comportement des instructions suivant

```
>>> U=[1, 2, 3]
>>> T=U
>>> T
[1, 2, 3]
```

```
>>> T.append(55)
>>> T
[1, 2, 3, 55]
>>> U
[1, 2, 3, 55]
```



On voix bien si on modifie T, on modifie U.

LISTES

Copie d'une liste simple :

Alors pour copie une liste simple il faut utiliser le tranchage (slicing) ou bien on utilise la méthode copy du module **copy**:

```
>>> U=[10, 3, 45, 1]
>>> T=U[:]
>>> T
[10, 3, 45, 1]
>>> T.append(55)
>>> T
[10, 3, 45, 1, 55]
>>> U
[10, 3, 45, 1]
```

```
>>> from copy import copy
>>> U=[10, 3, 45, 1]
>>> T=copy(U)
>>> T
[10, 3, 45, 1]
>>> T.append(55)
>>> T
[10, 3, 45, 1, 55]
>>> U
[10, 3, 45, 1]
```

LISTES

Liste des listes :

On utilisera couramment des listes qui contiennent des listes, en particulier pour représenter des matrices. L'accès aux éléments se fait de manière similaire :

```
>>> A=[[10, 3, 45], [1, 4, 8], [22, 6, 11]]

>>> A[0][0]
10

>>> A[0]
[10, 3, 45]

>>> A[2][1]=0
>>> A

[[10, 3, 45], [1, 4, 8], [22, 0, 11]]
```

LISTES

Exercice 8:

Ecrire une fonction **matrice*(n,m)*** qui retourne une matrice de taille nxm contenant des valeur aléatoires.

```
>>> matrice(2,3)
[[4, 6, 10],
 [11, 2, 0]]
```

```
def matrice(n,m):
    M=[]
    for i in range(n):
        L=[randint(-5,5) for j in range(m)]
        M.append(L)
    return M
```

```
def matrice1(n,m):
    return [[randint(-5,5) for i in range(n)] for j in range(m)]
```

LISTES

Copier Liste des listes :

Pour copie une liste de liste on utilise deux méthodes :

- On utilise la méthode **deepcopy** du module ***Copy***

```
>>> from copy import deepcopy
>>> A=[[10, 3, 45], [1, 4, 8], [22, 6, 11]]
>>> C=deepcopy(A)
>>> C
[[10, 3, 45], [1, 4, 8], [22, 6, 11]]
>>> A[2][1]=0
>>> A
[[10, 3, 45], [1, 4, 8], [22, 0, 11]]
>>> C
[[10, 3, 45], [1, 4, 8], [22, 6, 11]]
```

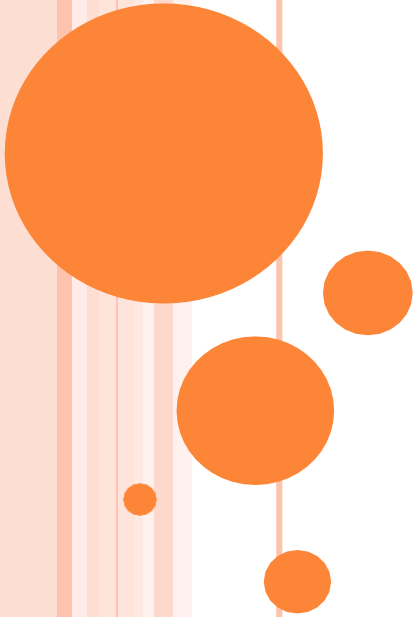
LISTES

Copier Liste des listes :

- On crée une nouvelle liste comme suit :

```
>>> A=[[10, 3, 45], [1, 4, 8], [22, 6, 11]]
>>> C=[ B[:] for B in A ]
>>> C
[[10, 3, 45], [1, 4, 8], [22, 6, 11]]
>>> A[2][1]=0
>>> A
[[10, 3, 45], [1, 4, 8], [22, 0, 11]]
>>> C
[[10, 3, 45], [1, 4, 8], [22, 6, 11]]
```

Tuple



TUPLES

- **Tuple**:
  - séquence contentant une suite d'éléments indexés à partir de 0 et délimités par des parenthèse :

```
>>> x = (‘Red', ‘Green', ‘Blue')
>>> print (x[2])
Blue
>>> y = 1, 9, 2
>>> print (y )
(1, 9, 2)
```

Initialisation

T=() | tuple() | (1,) | 'a', 'b', 'c', 'd' | ('a', 'b', 'c', 'd')

TUPLES

- **Tuples sont non mutable :**
  - **Une fois crée, on ne peut pas modifié la séquence**

```
>>> z = (5, 4, 3)
>>> z[2] = 0
Traceback: 'tuple' object does not
 support item assignment
```

  - **Les tuples support le slicing**

```
>>> z = (5, 4, 3, 2, 0)
>>> z[1:3]
(4, 3)
>>> z[-3:-1]
(3, 2)
```

TUPLES

- **Modifier les valeurs des tuples**

Une fois un tuple créé, vous ne pouvez pas modifier ses valeurs.

- Mais il existe des **solutions de contournement**. Vous pouvez convertir le tuple en liste, modifier la liste et reconvertir la liste en tuple.

```
>>> x =("apple“,"banana“,"cherry")
>>> y = list(x)
>>> y[1] = "kiwi“
>>> x = tuple(y)
>>> print(x)
("apple“,“kiwi“,"cherry")
```

```
x =("apple","banana","cherry")
x=x[:1]+("Kiwi",)+x[2:]
print(x)
```

TUPLES

- **Boucle à travers un tuple**

Vous pouvez parcourir les éléments du tuple en utilisant une boucle for.

```
>>> t= ("apple", "banana", "cherry")
>>> for x in t :  print(x)
 apple
 banana
 cherry
```

- **Test d'appartenance :**

Pour déterminer si un élément spécifié est présent dans un tuple, utilisez le mot clé **in** :

```
>>> t= ("apple", "banana", "cherry")
>>> 'kiwi' in t
 False
 >>> 'banana' in t
 True
```

TUPLES

- **Longueur du tuple**

Pour déterminer le nombre d'éléments d'un tuple, utilisez la méthode len() :

```
>>> t= ("apple", "banana", "cherry")
>>> len(t)
 3
```

- **Créer un tuple avec un seul élément**

Pour créer un tuple avec un seul élément, vous devez ajouter une virgule après l'élément, sauf si Python ne reconnaîtra pas la variable comme un tuple.

```
>>> t= (2)
>>> type(t)
 <class 'int'>
 >>> t= (2,)
 <class ‘tuple'>
```

TUPLES

- **Table de fonctions**:

Les Principales opérations et méthodes applicables à un tuple

```
len(U)                    # nombre de coordonnées
U+V                       # retourne la concaténation
n * U   ou   U * n        # concaténation répétée n fois
U.count(a)                # retourne le nombre d'occurrences de a
U.index(a)                # retourne le premier indice de a, ou une
                          # erreur si a n'est pas un attribut
a in U                    # teste l'appartenance de a à U
```

N.B : une fonction qui retourne plusieurs valeurs normalement retourne un tuple