L'essentiel en une 1/2-page

```python
# Liste de listes
Liste=[[1,2],[3,4]]
for sous_liste in Liste:
    print(sous_liste)

for sous_liste in Liste:
    for x in sous_liste:
        print(x)

N=len(Liste)
for i in range(N):
    for j in range(N):
        print(Liste[i][j])

# Un exemple mathématique
# Calcul de $\sum_{i=1}^{n}\sum_{j=i}^{n}(i+j)$
def somme_double(n):
    S=0
    for i in range(1,n+1):
        for j in range(i,n+1):
            S+=i+j
    return S

# Tableaux numpy
import numpy as np
T=np.array([[1,2],[3,4]])
print(2*T)
print(T*T)
print(np.dot(T,T))

N=len(T)
for i in range(N):
    for j in range(N):
        print(T[i,j])


U=np.array([[4,3],[2,1]])
V=np.dot(T,U) # produit matriciel
# Attention, T*U : produit coefficient par coefficient
```

Exercice 1: (Solution)

On appelle distance entre deux nombres réels $(x, y) \in \mathbb{R}^2$ la quantité $|x-y|$ que l'on obtient avec la commande `abs(x-y)` en `Python`.

1. Écrire une fonction `distMin(L)` renvoyant la plus petite distance entre deux éléments d'une liste `L` de nombres.

   Par exemple `distMin([1,3,5,10])` renvoie 2 et `distMin([1,3,5,10,1])` renvoie 0.
2. Faire de même lorsqu'on remplace `L` par `T` une liste de listes de mêmes longueurs.

   On appellera cette fonction `distMin1(T)`.

   Par exemple `distMin1([[1,5],[3,4]])` renvoie 1.
3. Modifier la fonction de la question précédente afin qu'elle renvoie le couple des valeurs les plus proches du tableau `T` et pas leur distance. On appellera cette fonction `val_prox(T)`

   Par exemple `val_prox([[1,5],[3,4]]` renvoie `(5,4)`.

   Lorsqu'il y a plusieurs couples dont la distance est minimale, la fonction renverra la liste de ces couples. Par exemple, `val_prox([[5,4],[3,1]])` renverra `[(5,4),(4,3)]`.

Exercice 2: (Solution)

L'objectif de cet exercice est d'écrire une fonction `recherche(phrase,facteur)` dont les arguments sont deux chaînes de caractères et renvoyant un booléen indiquant si la chaîne `facteur` appartient à la chaîne `phrase`.

Par exemple, `recherche("PTSIDumont","SI")` renvoie `True` tandis que `recherche("PTSIDumont","SImon")` renvoie `False`.

L'idée est d'utiliser deux indices `i` et `j` et de comparer le caractère `facteur[j]` avec le caractère `phrase[i+j]`.

- On utilisera une première boucle `for` et un premier indice `i`.
- On utilisera également une seconde boucle `while` imbriquée dans cette boucle `for`. Dans cette boucle `while`, on utilise un second indice `j`, initialement nul, que l'on incrémente à chaque itération. Cet indice va permettre de parcourir tous les caractères du mot `facteur`.

  On compare alors `facteur[j]` et `phrase[i+j]`.
- Si `facteur[j]!=phrase[i+j]` inutile d'incrémenter l'indice `j`.

  On passe à l'indice `i` suivant.
- Si `facteur[j]=phrase[i+j]`. On incrémente `j+=1` et on compare les caractères suivants de `facteur` et `phrase`.

- On continue tant que `phrase[i+j]=facteur[j]`. On s'arrête alors :
  - Soit lorsque la valeur de `j=len(facteur)`. Dans ce cas, on a découvert `facteur` dans `phrase`.
  - Soit lorsque l'égalité `phrase[i+j]=facteur[j]` n'est plus vérifiée : on passe à l'indice `i` suivant.

On reprend les deux exemples `recherche("PTSIDumont","SI")` et `recherche("PTSIDumont","SImon")`.

L'évolution des indices `i,j` est présentée dans un tableau à double entrée. Chaque cellule du tableau est le booléen `phrase[i+j]==facteur[j]`.

Le premier exemple `recherche("PTSIDumont","SI")`

	facteur	S	I
Phrase	i \ j	0	1
P	0	False	
T	1	False	
S	2	True	True
I	3	Fin de la boucle for	
D	4		
u	5		
m	6		
o	7		
n	8		
t	9		

$(i,j)=(0,0)$: `phrase[0]!=facteur[0]`. Inutile d'incrémenter j.

$(i,j)=(1,0)$: `phrase[1]!=facteur[0]`. Inutile d'incrémenter j.

$(i,j)=(2,0)$: `phrase[2]=facteur[0]`. On incrémente `j+=1`
$\hookrightarrow$ $(i,j)=(2,1)$:`phrase[2+1]=facteur[1]`.
Le facteur `SI` appartient bien à `"PTSIDumont"`.

Le second exemple `recherche("PTSIDumont","SImon")`.

Notez qu'il n'est pas nécessaire d'itérer sur l'indice `i` au délà de `len(phrase)-len(facteur)`.

	facteur	S	I	m	o	n
Phrase	i \ j	0	1	2	3	4
P	0	False				
T	1	False				
S	2	True	True	False		
I	3	False				
D	4	False				
u	5	False				
m	6	Fin de la boucle for				
o	7					
n	8					
t	9					

Phrase[i+j]==Facteur[j]

Écrire la fonction `recherche(phrase,facteur)`.

Exercice 3: (Solution)

Dans cet exercice on propose d'écrire une fonction `tri_insertion(L)` d'argument une liste d'entiers et renvoyant la liste triée dans l'ordre croissant.
Le principe de l'insertion est celui que l'on utilise en général pour classer, par exemple, des livres sur une étagère (par ordre alphabétique d'auteur).
On range le premier livre sur l'étagère, puis on place le deuxième livre à sa place en respectant l'ordre alphabétique. On insère ensuite le troisième élément en parcourant la liste des livres déjà rangés et en la plaçant à sa bonne place. Et, ainsi de suite.

1. Écrire une fonction `insere(L,x)` d'arguments une liste d'entiers **triée dans l'ordre croissant** et un entier `x` et renvoyant une liste composée des mêmes éléments que `L` plus l'entier `x` placé à la bonne place.
   Par exemple, `insere([1,2,4],3)` renvoie `[1,2,3,4]`.
2. Écrire la fonction `tri_insertion(L)`.
3. En utilisant la fonction `time` et en générant des listes `L` de nombres entiers dans l'intervalle $j \in [\![1, 1000]\!]$, évaluer le temps de calcul nécessaire à l'exécution de la fonction `tri_insertion(L)` en fonction de la taille de la liste `L`. On fera évoluer cette taille entre 1 et 1000.

On représentera graphiquement l'évolution du temps de calcul de la fonction `tri_insertion(L)` en fonction du nombre d'éléments à trier. Voir en **ANNEXE** le fonctionnement des modules `time`, `matplotlib`, `random`.

Exercice 4: (Solution)

Dans cet exercice on propose une méthode de tri d'une liste composée de nombres entiers appelée **tri à bulle**.
L'idée est de parcourir la liste de gauche à droite et de comparer les éléments consécutifs : on les permute s'ils ne sont pas dans l'ordre croissant.
Au cours des itérations successives de l'algorithme, les plus grands éléments sont décalés vers la droite (ils sont poussés vers la surface comme des bulles).
Regardons ce que cette idée donne sur un exemple.

PREMIER PASSAGE

`L=[45,3,50,40]`

Lors de ce premier passage, on commence par comparer 45 et 3 : on les permute, la liste devient :

`L=[3,45,50,40]`

Toujours lors de ce premier passage, on compare 45 et 50 : on ne fait rien, la liste est toujours la suivante :

`L=[3,45,50,40]`

Attention, le plus grand élément détecté à ce stade est 50 et c'est ce nombre qui va être comparé à l'élément suivant.
A la fin de ce premier passage, on compare et on permute 50 et 40. La liste devient :

`L=[3,45,40,50]`

DEUXIÈME PASSAGE

Il y a encore des permutations à effectuer, on passe à l'itération suivante : il ne sera pas nécessaire de parcourir toute la liste. En effet, le plus grand élément (c'est-à-dire 50) a été déplacé tout à droite (à la surface) : lors de ce deuxième passage, on ne parcourt que les 3 premiers éléments de la liste.
Lors de ce passage, on va comparer 3 à 45, puis 45 à 40 pour obtenir finalement

`L=[3,40,45,50].`

TROISIÈME PASSAGE

La dernière itération consistera simplement à comparer 3 et 40.

1. Écrire une fonction `tri_bulle(L)` d'argument une liste d'entiers et renvoyant la liste triée dans l'ordre croissant.
2. En utilisant la fonction `time` et en générant des liste de nombres entiers aléatoires de taille $j \in [\![1, 1000]\!]$ évaluer le temps de calcul nécessaire à l'exécution de la fonction `tri_bulle(L)` en fonction de la taille de la liste `L`.

Exercice 5: (Solution)

Dans cet exercice on travaillera avec des tableaux `numpy`.

1. Écrire une fonction `puissance(A,n)` d'argument une matrice carrée et un entier $n \in \mathbb{N}$ et renvoyant la puissance A^n. On n'utilisera pas la fonction `power`.
2. Un feu bicolore, lorsqu'il est rouge à un instant donné, passe au vert à l'instant suivant avec la probabilité p et s'il est vert, il passe au rouge avec probabilité q.
   On suppose que $p, q \in]0;1[$. On note r_n (resp. v_n) la probabilité que ce feu soit rouge (resp. vert) à l'instant $t = n$. On suppose que $r_0 + v_0 = 1$.
   (a) Montrer mathématiquement qu'il existe une matrice $A \in \mathscr{M}_2(\mathbb{R})$ telle que :
   $$\forall n \in \mathbb{N}, \begin{pmatrix} r_{n+1} \\ v_{n+1} \end{pmatrix} = A \begin{pmatrix} r_n \\ v_n \end{pmatrix}.$$
   (b) Montrer que
   $$\begin{pmatrix} r_n \\ v_n \end{pmatrix} = A^n \begin{pmatrix} r_0 \\ v_0 \end{pmatrix}$$
   et en déduire une approximation des probabilités r_n et v_n à l'aide de Python.

ANNEXE (Exercices 3 - 4)

```python
import time
import random # générateur de hasard
import matplotlib.pyplot as plt
# bibliothèque de représentation graphique

# 1)
# Construction des listes des abscisses / Ordonnées
Abs=list(range(1,1000)) # [1,2, ... ,1000]
Ord=[] # a compléter : temps de calculs

# 2)
# Construction d'une liste aléatoire d'entiers
L=[]
for i in range(1000):
    L.append(rd.randint(1,1000))
    # rd.randint(1,1000) renvoie un entier 1 ⩽ p ⩽ 1000
    # len(L)=1000

# 3)
# Calcul du temps d'éxécution de tri(L[0:i])
    t=time.time()
    tri(L[0:i]) # tri_insertion ou tri_bulle
    t=time.time()-t
    # t=temps de calcul de tri(L[0:i])
    Ord.append(t)

plt.plot(Abs,Ord,'.')
# construit le graphe des temps de calculs
# en fonction de la longueur de L[0:i]
#'.b' pour un nuage de points bleus
# '.r' pour un nuage de points rouges

plt.show() # affiche le graphe
```