Term Nsi Evaluation pratique 1 – CORRIGE-

Se loguer avec l'identifiant : **exam02.eleve** Mot de passe :

Ce DS est composé de 2 parties complètement indépendantes. Dans chaque partie, les questions peuvent être souvent traitées indépendamment.

Pour lancer PYZO, aller dans le répertoire ***Examens(Z :) /exam02/sujets*** et double-cliquer sur le raccourci :

Pyzo_win10_STI.cmd

Vous trouverez également dans ce répertoire, les fichiers suivants, **qu'il s'agit dès à présent de copier et coller sur un répertoire perso à créer sur le bureau de l'ordinateur** :

- *sujet.pdf* contient le sujet que vous êtes en train de lire,
- *poudlard.csv* contient 50 lignes de données qui seront utilisées dans l'exercice 2 ,
- *exercice2.py* contient le code python de 4 fonctions qui seront utilisées dans l'exercice 2.

PARTIE 1. : RECHERCHE D'UNE VALEUR DANS UNE LISTE

Les codes demandés dans cette partie seront écrits dans un fichier que vous nommerez *exercice1_votreNom.py* à créer dans votre répertoire de travail sur le bureau de l'ordinateur. **Ce n'est qu'un fin d'exercice** que vous copierez ce fichier dans l'espace de dépôt : ***Examens(Z :) /exam02/copies***

Question 1.: Compléter le code python de la fonction *rechercheNaive()* donnée ci-contre. Ses paramètres sont une liste ℓ de nombres **déjà triés par ordre croissant** et une valeur numérique val.

```
def rechercheNaive(l,val) :
    indices = []
```

Cette fonction retourne la liste des indices i de ℓ pour lesquels $\ell\,[\,i\,]$ est égale à val . Vous réaliserez un simple parcours de liste que vous pourrez écourter car on sait ici que la liste ℓ est triée.

On donne ci-dessous 3 exécutions tests réalisées dans la console :

```
>>> rechercheNaive([-74,14,14,14,14,34,49,57,98],14)
[1, 2, 3, 4]

>>> rechercheNaive([-74,14,14,14,14,34,49,57,98],-74)
[0]

>>> rechercheNaive([-74,14,14,14,14,34,49,57,98],15)
[]
```

CORRIGE

```
def rechercheNaive(l,val) :
    indices = []
    for i in range(len(l)) :
        if l[i] == val : indices.append(i)
        if l[i] > val : return indices
    return indices
```

Question 2.: Donner la classe de complexité de votre script (répondre sur le document réponse distribué)

Ce script a une classe de complexité $\mathcal{O}(n)$ car on réalise au pire un parcours entier de la liste.

Question 3.: Compléter le code python de la fonction *indiceVoisinMemeValeur()* donnée ci-contre. Ses paramètres sont une liste ℓ de nombres déjà triés par ordre croissant et une valeur numérique $0 \leq ind < len(\ell)$

```
def indiceVoisinMemeValeur(l,ind) :
    indices = [ind]
    i = ind + 1
    while i < len(l) and l[i]==l[ind] :
        indices.append(i)
```

Cette fonction retourne la liste des indices i de ℓ pour lesquels l'élément $\ell\,[\,i\,]$ est égale à $\ell\,[\,ind\,]$. Vous réaliserez un parcours partiel sur les voisins de droite jusqu'à ce que $\ell\,[\,i\,] \neq \ell\,[\,ind\,]$, puis même chose sur les voisins de gauche.

On donne ci-dessous 3 exécutions tests réalisées dans la console :

```
>>> indiceVoisinMemeValeur([-74,14,14,14,14,34,49,57,98],2)
[2, 3, 4, 1]

>>> indiceVoisinMemeValeur([-74,14,14,14,14,34,49,57,98],0)
[0]

>>> indiceVoisinMemeValeur([-74,14,14,14,14,34,49,57,98],7)
[7]
```

CORRIGE

```
def indiceVoisinMemeValeur(l,ind) :
    indices = [ind]
    i = ind + 1
    while i < len(l) and l[i]==l[ind] :
        indices.append(i)
        i = i + 1
    i = ind - 1
    while i >= 0 and l[i]==l[ind] :
        indices.append(i)
        i = i - 1
    return indices
```

Question 4.: On donne ci-contre le script python de la fonction rechercheDichotomique().

```
def rechercheDichotomique(l,val) :
    iD = 0
    iF = len(l)-1
    while iD <= iF :
        iM = (iD + iF)//2
        if l[iM] == val : return iM
        elif val > l[iM] : iD = iM + 1
        elif val < l[iM] : iF = iM - 1
    return False
```

⇨ Compléter sur le document réponse, le tableau ci-dessous, donnant les valeurs prises par les variables identifiées, en faisant fonctionner l'algorithme « à la main », pour l'exécution suivante :

```
rechercheDichotomique([-74,14,14,14,14,34,49,57,98],57)
```

iD	iF	iM	$\ell[iM]$	
0	8	4	14	[-74,14,14,14,14,34,49,57,98]
5	8	6	49	[-74,14,14,14,14,34,49,57,98]
7	8	7	57	[-74,14,14,14,14,34,49,57,98]

⇨ Refaire la même chose pour l'exécution suivante :

```
rechercheDichotomique([-74,14,14,14,14,34,49,57,98],14)
```

iD	iF	iM	$\ell[iM]$	
0	8	4	14	[-74,14,14,14,14,34,49,57,98]
				[-74,14,14,14,14,34,49,57,98]
				[-74,14,14,14,14,34,49,57,98]

Question 5.: Modifier le script de cette fonction *rechercheDichotomique()* , en faisant appel à la fonction *indiceVoisinMemeValeur()* mise au point dans la question 3, afin que *rechercheDichotomique()* retourne à présent la liste des indices i de ℓ pour lesquels $\ell\,[\,i\,]$ est égale à val

On donne ci-dessous 3 exécutions tests réalisées dans la console :

```
>>> rechercheDichotomique([-74,14,14,14,14,34,49,57,98],57)
[7]

>>> rechercheDichotomique([-74,14,14,14,14,34,49,57,98],11)
[]

>>> rechercheDichotomique([-74,14,14,14,14,34,49,57,98],14)
[4, 3, 2, 1]
```

CORRIGE

```
def rechercheDichotomique(l,val) :
    iD = 0
    iF = len(l)-1
    while iD <= iF :
        iM = (iD + iF)//2
        if l[iM] == val :
            indices = indiceVoisinMemeValeur(l,iM)
            return indices
        elif val > l[iM] : iD = iM + 1
        elif val < l[iM] : iF = iM - 1
    return []
```

Question 6.: Donner la classe de complexité de votre script (répondre sur le document réponse distribué)

Ce script a une classe de complexité $\mathcal{O}(log_2(n))$ car on divise la zone d'étude par 2 à chaque itération. Le script de la fonction *indiceVoisinMemeValeur()* ne rajoute pas de complexité car les valeurs identiques de la liste sont normalement peu nombreuses. Mais dans le pire des cas, si toutes les valeurs de la liste sont égales à celle recherchée, la complexité est $\mathcal{O}(log_2(n) + n)$. Sachant que le nombre $log_2(n)$ est bien inférieur à n lorsque la taille de la liste est importante, on peut dire que dans le pire des cas, la classe de complexité est $\mathcal{O}(n)$.

⇨ Copier votre fichier *exercice1_votreNom.py* dans l'espace de dépôt : ***Examens(Z :) /exam02/copies***

PARTIES 2. : ALGORITHME DES K PLUS PROCHES VOISINS

Présentation :

A l'entrée à l'école de Poudlard, le Choixpeau magique répartit les élèves dans les différentes maisons (*Gryffondor*, *Serpentard*, *Serdaigle* et *Poutsouffle*) en fonction de leur courage, leur loyauté, leur sagesse et leur malice.

L'objectif de cet exercice est de réaliser un code qui permet de répartir un nouvel élève dans une des 4 maisons, en fonction des informations ci-dessous, qui concernent les élèves déjà inscrits à l'école et qui se sont vus attribuer une note sur 10 pour chacune de leur vertu :

Nom	***Courage***	***Loyauté***	***Sagesse***	***Malice***	***Maison***
Abdenour	9	4	7	10	Serpentard
Titouan	9	3	4	7	Gryffondor

Ilian	10	6	5	9	Gryffondor
Mathis	2	8	8	3	Serdaigle
Félix	10	4	2	5	Gryffondor
Corentin	10	4	9	9	Poufsouffle
Kylian	10	7	4	7	Gryffondor
etc ….					

La liste des élèves déjà répartis dans l'école est beaucoup plus longue. On en a ci-dessus qu'un petit extrait. La liste entière est contenue dans le fichier *poudlard.csv* mis à disposition.

Une nouvelle élève arrive à l'école de Poudlard. Elle est suédoise et s'appelle *Camilla*. Un test de positionnement lui a permis d'obtenir une note de 8 en Courage, 6 en Loyauté, 6 en Sagesse, 6 en Malice.

L'objectif de l'exercice est de réaliser un code qui reprend un algorithme du type « *k plus proches voisins* » et qui déterminera, parmi les élèves déjà inscrits à Poudlard, les maisons des k plus proches voisins de *Camilla*.

Avant de se lancer dans la réalisation de ce code, on donne des indications sur la distance à utiliser dans cet algorithme …

On décide d'utiliser la relation suivante pour calculer la distance entre deux élèves 1 et 2 :

$$d = |courage1 - courage2| + |loyauté1 - loyauté2| + |sagesse1 - sagesse2| + |malice1 - malice2|$$

Par exemple la distance d entre Camilla la suédoise et Abdenour sera :

Nom	***Courage***	***Loyauté***	***Sagesse***	***Malice***	***Maison***
Abdenour	9	4	7	10	Serpentard
Camilla	8	6	6	6	?

$$d = |9 - 8| + |4 - 6| + |7 - 6| + |10 - 6|$$
$$d = |1| + |-2| + |1| + |4|$$
$$d = 1 + 2 + 1 + 4 = 8$$

En langage python, la fonction qui calcule la valeur absolue d'un nombre est ***abs()*** .

Par exemple :

```
>>> abs(-2)
2

>>> abs(4-100)
96

>>> abs(40-1)
39
```

Travail à réaliser :

⇨ Modifier le nom du fichier *exercice2.py* qui a déjà été copié dans votre dossier de travail, sur le bureau de l'ordinateur. Le renommer *exercice2_votreNom.py*

Ce fichier contient les fonctions suivantes :

- *lectureDonnees()* : voir le docString écrit en commentaire dans le fichier .py . Exécution test dans la console :

```
>>> donnees = lectureDonnees("poudlard.csv")

>>> donnees[36]
['Pansy', 4.0, 4.0, 10.0, 8.0, 'Serpentar']
```

- *echange()* : voir le docString écrit en commentaire dans le fichier .py . Exécution test dans la console :

```
>>> l = [10,11,12,13,14]

>>> echange(l,1,3)
True

>>> l
[10, 13, 12, 11, 14]
```

- *triSelection()* : voir le docString écrit en commentaire dans le fichier .py . Exécution test dans la console :

```
>>> triSelection([4,3,-8,0])
[-8, 0, 3, 4]
```

- *triInsertion()* : voir le docString écrit en commentaire dans le fichier .py . Exécution test dans la console :

```
>>> triInsertion([4,3,-8,0])
[-8, 0, 3, 4]
```

Question 1.: Compléter le code python de la fonction *distance()* , ébauchée et documentée ci-dessous. Elle retourne la distance entre les 2 listes en paramètres.

```
def distance(lA , lB):
    '''
        Paramètres :
            lA : liste du type  [Nom (string) , Courage(int) , Loyaute(int) , Sagesse(int) , Malice(int), Maison(string) ]
            lB : liste du type  [Nom (string) , Courage(int) , Loyaute(int) , Sagesse(int) , Malice(int) ]
        Renvoie la distance (float) entre "lA" et "lB" : distance = |courage1-courage2| +
        |loyaute1-loyaute2| + |sagesse1-sagesse2| + |malice1-malice2|
    '''
```

Pour validation, on donne ci-dessous un exemple test exécuté dans la console :

```
>>> distance( ["Abdenour",9,4,7,10,"Serpentard"],["Camilla" , 8 , 6 , 6 , 6 ])
8
```

CORRIGE

```
def distance(lA , lB):
    d = 0
    for i in range(1,5) :
        d = d + abs(lA[i]-lB[i])
    return d
```

Question 2.: Modifier le code de la fonction *triSelection()*, afin qu'il puisse trier une liste dont chaque élément est une sous-liste ℓ de 6 valeurs $[l[0]$, $l[1]$, $l[2]$, $l[3]$, $l[4]$, $l[5]$, $l[6]$]. Ces sous-listes sont triés par ordre croissant de la valeur $l[6]$. Le principe du tri étant « *par sélection* », on trie uniquement les k sous-listes qui possèdent les valeurs $l[6]$ les plus petites. La variable k est le second paramètre de cette fonction.

Pour validation, on donne ci-dessous un exemple test exécuté dans la console pour une valeur k égale à 1 : `triSelection(l,1)`

```
>>> l = [["a",1,2,3,4,"Serpentar",77] , ["b",1,2,3,4,"Griffondor",11] , ["b",1,2,3,4,"Griffondor",25]]

>>> triSelection(l,1)
[['b', 1, 2, 3, 4, 'Griffondor', 11], ['a', 1, 2, 3, 4, 'Serpentar', 77], ['b', 1, 2, 3, 4, 'Griffondor', 25]]
```

CORRIGE

```
def triSelection(liste,k) :
    n = len(liste)
    for i in range(k) :
        jMin = i
        for j in range(i+1,n) :
            if liste[j][-1]<liste[jMin][-1] :
                jMin = j
        echange(liste,i,jMin)
    return liste
```

Question 3.: Créer le code de la fonction *ecritureConsole()*, qui prend en paramètres une liste de sous-listes ℓ de 6 valeurs $[l[0] , l[1] , l[2] , l[3] , l[4] , l[5] , l[6]]$, une seconde liste simple et un nombre k. Cette fonction retourne un string qui donne, pour les k premiers éléments de cette liste de sous-listes, la valeur $l[5]$ (*string*) qui revient le plus souvent. Pour validation, on donne ci-dessous un exemple test exécuté dans la console pour une valeur k égale à 1 :

```
>>> l = [["a",1,2,3,4,"Serpentar",77],["b",1,2,3,4,"Griffondor",11],["c",1,2,3,4,"Griffondor",25]]

>>> retour = ecritureConsole(l,["Camilla",8,6,6,6],3)

>>> print(retour)

Nombres de voisins Griffondor : 2
Nombres de voisins Serpentar : 1
Nombres de voisins Serdaigle : 0
Nombres de voisins Poufsouffle : 0

Camilla sera dans la famille : Griffondor
```

CORRIGE

```
def ecritureConsole(donnees,inconnue,k) :
    dic = {"Griffondor":0 ,"Serpentar":0 , "Serdaigle":0 , "Poufsouffle":0}
    for i in range(k) :
        nom = donnees[i][5]
        dic[nom] = dic[nom] + 1
    cleMax = "Griffondor"
    if dic[cleMax] < dic["Serpentar"] : cleMax = "Serpentar"
    if dic[cleMax] < dic["Serdaigle"] : cleMax = "Serdaigle"
    if dic[cleMax] < dic["Poufsouffle"] : cleMax = "Poufsouffle"
    sortie = cleMax
    for cle in dic :
        if dic[cle] == dic[cleMax] and cle != cleMax :
            sortie = sortie + " ou " + cle


    resultat =f"""
Nombres de voisins Griffondor : {dic['Griffondor']}
Nombres de voisins Serpentar : {dic['Serpentar']}
Nombres de voisins Serdaigle : {dic['Serdaigle']}
Nombres de voisins Poufsouffle : {dic['Poufsouffle']}

{inconnue[0]} sera dans la famille : {sortie}
"""
    return resultat
```

Question 4.: Compléter le code python de la fonction *kVoisins()* , ébauchée et documentée ci-dessous. Il permet d'afficher dans la console les k plus proches voisins de Camilla :

```
def kVoisins(inconnue,k) :
    """
    Paramètres :inconnue, liste contenant les attibuts de la personne inconnue
                k (int), nombre des plus proches voisins
    Sortie : aucun renvoi
    """
```

On exécutera entre autres dans cette fonction, les fonctions *distance()* , *triSelection()* et *ecritureConsole()* , mises au point dans les questions précédentes.

Pour validation, on donne le résultat dans la console, de l'exécution du programme principal ci-contre ($k = 2$) :

```
# Main
donnees = lectureDonnees("poudlard.csv")
kVoisins(["Camilla",8,6,6,6],2)
```

```
>>> (executing file "exercice2.py")

Nombres de voisins Griffondor : 1
Nombres de voisins Serpentar : 1
Nombres de voisins Serdaigle : 0
Nombres de voisins Poufsouffle : 0

Camilla sera dans la famille : Griffondor ou Serpentar
```

Si la valeur de 4 est de 4, on obtient le résultat suivant :

```
>>> (executing file "exercice2.py")

Nombres de voisins Griffondor : 2
Nombres de voisins Serpentar : 1
Nombres de voisins Serdaigle : 1
Nombres de voisins Poufsouffle : 0

Camilla sera dans la famille : Griffondor
```

CORRIGE

```
def kVoisins(inconnue,k) :
    """
    Paramètres :inconnue, liste contenant les attibuts de la personne inconnue
                k (int), nombre des plus proches voisins
    Sortie : aucun renvoi
    """
    for i in range(len(donnees)) :
        d = distance(donnees[i],inconnue)
        donnees[i].append(d)
    triSelection(donnees,k)
    resultat = ecritureConsole(donnees,inconnue,k)
    print(resultat)

# Main
donnees = lectureDonnees("poudlard.csv")
kVoisins(["Camilla",8,6,6,6],2)
```

Question 5.(plus difficile) : Modifier le code python donné de la fonction *triInsertion()* , afin de pouvoir l'utiliser à la place de celui de la fonction triSelection(). Avec ce type de tri, on ne peut par contre, pas trier partiellement la liste. Il est nécessaire de la trier entièrement. Tester le code afin d'obtenir le même résultat que celui donné juste avant.

CORRIGE

```
def triInsertion(liste) :
    """
    Paramètres: liste (list) : liste
    Sortie : liste (list) : même liste, mais triée
    """
    n = len(liste)
    for i in range(1,n) :
        val = liste[i]
        j = i-1
        while liste[j][-1] > val[-1] and j >=0 :
            liste[j+1] = liste[j]
            j = j - 1
        liste[j+1] = val
```

⇨ Copier votre fichier *exercice2_votreNom.py* dans l'espace de dépôt : ***Examens(Z :) /exam02/copies***