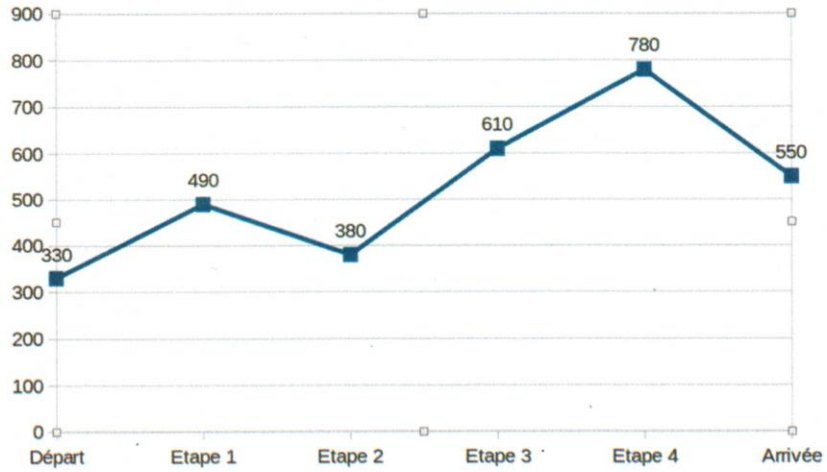
Spécialité NSI Première	**DEVOIR SURVEILLE DE**	Mardi 17 mai 2022
Lycée d'Avesnières	**NSI**	Durée : 1 h
Année scolaire 2021-2022	**N° 8**	Calculatrice interdite

L'énoncé complet est à rendre avec la copie. Répondez directement sur l'énoncé.

NOM : ..... **Prénom :** .....

Exercice 1 (3 points)

Le dénivelé cumulé positif d'une course de montagne est la somme totale des dénivelés de l'ensemble des ascensions durant la course.



Sur l'exemple ci-dessus :

- la course commence par une ascension de dénivelé positif 160 (490−330)
- entre l'étape 2 et l'étape 3, le dénivelé positif est de 230 (610−380)
- entre l'étape 3 et l'étape 4, le dénivelé positif est de 170 (780−610)
- les autres parties de la course sont des descentes

Le dénivelé cumulé positif total de cette course est donc 160+230+170=560
Écrire une fonction denivele_positif qui prend en argument la liste non vide des altitudes atteintes à la fin de chaque ascension et de chaque descente pendant la course et qui renvoie le dénivelé cumulé positif de cette course.

```
def denivele_positif(altitudes):
    cumul = 0
    for i in range(len(altitudes)-1):
        delta = altitudes[i+1] - altitudes[i]
        if delta > 0:
            cumul = cumul + delta
    return cumul

# tests
assert denivele_positif([330, 490, 380, 610, 780, 550]) == 560
assert denivele_positif([200, 300, 100]) == 100
```

Exercice 2 (7 points)

Augustin a déjà eu, au cours de ce trimestre, trois notes sur 20 en NSI :

- 13,5 avec un coefficient 1
- 15 avec un coefficient 2
- 16,5 avec un coefficient 4.

En révisant pour la 4e et dernière évaluation du trimestre qui aura un coefficient 4, il veut prévoir sa moyenne trimestrielle de NSI en fonction de sa dernière note. Sa dernière note et sa moyenne trimestrielle peuvent être des nombres à virgule. Pour cela, il écrit la fonction :

```
def moyenne(last):
    ...

    return moy
```

qui prend en argument sa dernière note et qui renvoie sa moyenne trimestrielle.

1) Écrire sous la forme d'une *docstring* le prototypage de cette fonction.

```
"""
Calcule la moyenne trimestrielle en fonction de la
dernière note du trimestre.
Paramètre : last de type flottant.
            C'est la note obtenue à la dernière évaluation
            du trimestre.
Renvoie :   moy de type flottant.
            C'est la moyenne calculée avec toutes les notes.
"""
```

2) Décrire la précondition sur l'argument et la postcondition sur le résultat.

- En précondition : last doit être supérieure ou égale à 0 et inférieure ou égale à 20.
- En postcondition : moy doit être supérieure ou égale à 0 et inférieure ou égale à 20.

3) Écrire la fonction moyenne(derniere_note) sur la copie **en ajoutant des assertions** pour les précondition et postcondition.

```
def moyenne(last):
    assert 0 <= last and last <= 20, "last doit être sur [0;20]"
    somme = 13.5*1 + 15*2 + 16.5*4 + last*4
    moy = somme / (1+2+4+4)
    assert 0 <= moy and moy <= 20, "moy doit être sur [0;20]"
    return moy
```

Exercice 3 (10 points)

L'objectif de cet exercice est d'implémenter en Python un algorithme kNN afin de déterminer la classe de deux iris observés en pleine nature, parmi les trois variétés suivantes :

Iris Setosa

Iris Versicolor

Iris Virginica

Pour classer les deux iris qu'on a observés en pleine nature, on retient comme critères :

- La largeur des sépales en cm
- La largeur des pétales en cm

Le tableau ci-dessous résume les critères pour les deux spécimens d'iris observés :

Numéro du spécimen	Largeur des sépales	Largeur des pétales
1	3,5	0,2
2	3,5	1,2

Document 1 : Fichiers (incomplets) déjà écrits au départ

- irisNum1.csv est le jeu de données avec 50 iris Setosa, 50 iris Versicolor et 50 iris Virginica.

```
id,sepal.length,sepal.width,petal.length,petal.width,variety
1,5.1,3.5,1.4,0.2,Setosa
2,4.9,3,1.4,0.2,Setosa
3,4.7,3.2,1.3,0.2,Setosa
4,4.6,3.1,1.5,0.2,Setosa
5,5,3.6,1.4,0.2,Setosa
...

149,6.2,3.4,5.4,2.3,Virginica
150,5.9,3,5.1,1.8,Virginica
```

- iris.py Programme permettant de classer un spécimen d'iris inconnu.

- iris.py Programme permettant de classer un spécimen d'iris inconnu.

```
import csv   #  Bibliotheque de fonctions pour manipuler des fichiers csv.
import matplotlib.pyplot as plt   #  Bibliotheque de fonctions pour graphiques.
import operator   #  Bibliotheque contenant la fonction itemgetter()
import math   #  Bibliotheque contenant la fonction sqrt()


def lecture_fichier(nom_fichier):

    """
    Importe un fichier csv existant et le transforme en liste de listes.

    Parametres : nom_fichier : du type chaine de caracteres.
    -----------
                 C'est le nom du fichier csv (encode en utf-8),
    Par exemple 'irisNum1.csv'


    Renvoie : Une liste de listes.
    ---------
              Chaque sous-liste est une ligne du fichier csv.

    Par exemple :
    [['id', 'sepal.length', 'sepal.width', 'petal.length', 'petal.width', 'variety'],\
     ['1', '5.1', '3.5', '1.4', '0.2', 'Setosa'],\
     ['2', '4.9', '3', '1.4', '0.2', 'Setosa'],\
     ['3', '4.7', '3.2', '1.3', '0.2', 'Setosa'], ...]

    """

    with open(nom_fichier, mode='r', encoding='utf-8-sig') as fichier_ouvert:
        return [ligne for ligne in csv.reader(fichier_ouvert)]
```

```
def creer_tuple_de_listes(L):

    """
    A partir d'une liste de listes representant un fichier csv,
    renvoie un tuple de 3 listes.

    Parametres :   L : du type tableau (liste de listes Python).
    ------------

    Par exemple :
    [['id', 'sepal.length', 'sepal.width', 'petal.length', 'petal.width', 'variety'],\
     ['1', '5.1', '3.5', '1.4', '0.2', 'Setosa'],\
     ['2', '4.9', '3', '1.4', '0.2', 'Setosa'],\
     ['3', '4.7', '3.2', '1.3', '0.2', 'Setosa'], ...]


    Renvoie :  (liste_sepal, liste_petal, liste_variete) du type tuple de 3 listes.
    ---------

    La 1ere liste contient la longueur (en type flottant) des sepales.
    La 2e liste contient la longueur (en type flottant) des petales.
    La 3e liste contient les varietes (en type chaine de caracteres).

    Par exemple :
    ([5.1, 4.9, 4.7, ...], [1.4, 1.4, 1.3, ...], ['Setosa', 'Setosa', 'Setosa', ...])

    """


    liste_sepal = [float(L[i][2]) for i in range(1, len(L))]
    liste_petal = [float(L[i][4]) for i in range(1, len(L))]
    liste_variete = [L[i][5] for i in range(1, len(L))]

    return (liste_sepal, liste_petal, liste_variete)
```

```
def affichePoints(x, y, variete):

    """
    A partir d'un tuple de 3 listes contenant les criteres 1 et 2 et l'etiquette,
    affiche un graphique de points colores.

    Parametres :   (x, y, variete) : Du type tuple de trois listes.
    ------------

    Par exemple :
    ([5.1, 4.9, 4.7, ...], [1.4, 1.4, 1.3, ...], ['Setosa', 'Setosa', 'Setosa', ...])

    """

    #  Creation des 150 points avec une couleur 'r' (red), 'b' (blue), 'g' (green)
    plt.plot(x[0:50], y[0:50], 'or', label='Setosa')  # 'or' (rond, rouge)
    plt.plot(x[50:100], y[50:100], 'ob', label='Versicolor')
    plt.plot(x[100:], y[100:], 'og', label='Virginica')

    #  Creation du titre, de la grille, des labels sur les axes et de la legende.
    plt.title("Fleurs d'iris")
    plt.grid()
    plt.xlabel('Largeur des sepales (cm)')
    plt.ylabel('Largeur des petales (cm)')
    plt.legend()

    #  Creation des 2 points specimens inconnus avec une couleur 'c' (cyan).
    plt.plot(3.5, 0.2, 'oc')
    plt.plot(3.5, 1.2, 'oc')

    #  Affichage du graphique.
    plt.show()
```

```
def distance(xA, yA, xB, yB):

    """
    Renvoie la distance euclidienne entre les points A et B.

    Parametres : xA, yA, xB, yB : de type flottant
    ------------
                Ce sont les coordonnees des points A et B.

    Par exemple xA vaut 3.0 ; yA vaut 5.0, xB vaut 4.0 ; yB vaut 6.0.


    Renvoie : d : de type flottant
    ---------
              C'est la distance AB.

    Par exemple :
    1.4142135623730951

    """
    d = math.sqrt((xB - xA)**2 + (yB - yA)**2)
    return d
```

```
def calcule_distances(xP, yP, liste):

    """
    Renvoie la liste des doublets (distance, classe)

    Parametres : xP, yP : de type flottant
    ------------
                Ce sont les coordonnees du point inconnu P a classer.

    Par exemple xP vaut 3.0 et yP vaut 1.8.

                liste : de type liste
                C'est le jeu de donnees.

    Par exemple :
    [[3.5, 0.2, 'Setosa'], [[3.5, 0.2, 'Setosa'], ...]


    Renvoie : liste_doublets : de type liste
    ---------
              C'est la liste des doublets
              [(distance entre P et le 1er elt, classe du 1er elt),\
              distance entre P et le 2e elt, classe du 2e elt), ...

    Par exemple :
    [(1.6763054614240211, 'Setosa'), (1.6, 'Setosa'), ...]

    """
    liste_doublets = []  # Initialisation de la liste des doublets.

    for i in range(len(liste)):
        d = distance(xP, yP, liste[i][0], liste[i][1])
        classe = liste[i][2]
        liste_doublets.append(d, classe)
    return liste_doublets
```

```
def tri(tableau):
    """


    Renvoie le tableau trie selon l'ordre des distances croissantes.

    Parametres : tableau : de type liste
    ------------
              C'est la liste des doublets non triee renvoyee par la
              fonction calcule_distances(xP, yP, liste).

    Par exemple tableau vaut :
    [(1.6763054614240211, 'Setosa'), (1.6, 'Setosa'), ...]


    Renvoie : liste_doublets_triee : de type liste
    ---------
             C'est la liste des doublets triee par ordre de distances croissantes.

    Par exemple liste_doublets_triee vaut :
    [(0.0, 'Virginica'), (0.0, 'Virginica'), ...]
    """
    liste_doublets_triee = sorted(tableau, key=operator.itemgetter(0))
    return liste_doublets_triee
```

La fonction sorted renvoie une nouvelle liste.

numéro de colonne selon laquelle on trie

Remarque :

Il est aussi possible de remplacer key=operator.itemgetter(0) par une fonction lambda : key=lambda x: x[0]

```
def classe(k, tableau_trie):

    """

    Renvoie le dictionnaire des classes: effectif.

    Parametres : k : de type entier. C'est le nombre de plus proches voisins.
    ------------
                 tableau_trie : de type liste
                 C'est la liste des doublets triee renvoyee par la fonction tri(tableau).

    Renvoie : dico_des_voisins : .de type dictionnaire
    ---------
              C'est le dictionnaire donnant les effectifs des classes parmi les k
              Proches voisins.

    Par exemple dico_des_voisins vaut :
    {'Setosa': 0, 'Versicolor': 2, 'Virginica': 11}

    """

    dico_des_voisins = {} # Initialisation du dictionnaire

    # On examine les k premiers doublets qui sont dans tableau_trie
    for i in range(k):
        classe = tableau_trie[i][1]
        # Si la classe est déjà dans le dictionnaire, on incrémente la valeur
        if classe in dico_des_voisins.keys():
            dico_des_voisins[classe] += 1
        # sinon on crée l'item avec la valeur 1.
        else:
            dico_des_voisins[classe] = 1

    return dico_des_voisins
```

Remarque:

Il y a plusieurs façons d'écrire le code de la fonction classe. Par exemple, utiliser dictionary.get(keyname, default_value). Ainsi, l'instruction conditionnelle if classe .... else .... peut être alors remplacée par:

```
dico_des_voisins[classe] = dico_des_voisins.get(classe, 0) + 1
```

```
#  Programme :

#   Construction de L qui vaut :
# [['id', 'sepal.length', 'sepal.width', 'petal.length', 'petal.width', 'variety'],\
# ['1', '5.1', '3.5', '1.4', '0.2', 'Setosa'],\
# ['2', '4.9', '3', '1.4', '0.2', 'Setosa'],\
# ['3', '4.7', '3.2', '1.3', '0.2', 'Setosa'], ...]
L = lecture_fichier("irisNum1.csv")


#   Construction du tuple de trois listes qui vaut :
#   ([5.1, 4.9, 4.7, ...], [1.4, 1.4, 1.3, ...], ['Setosa', 'Setosa', 'Setosa', ...])
(x, y, variete) = creer_tuple_de_listes(L)


#  Creation et affichage du graphique a partir du tuple
#  ([5.1, 4.9, 4.7, ...], [1.4, 1.4, 1.3, ...], ['Setosa', 'Setosa', 'Setosa', ...])
affichePoints(x, y, variete)


#  Construction d'une liste de la forme [[3.5, 0.2, 'Setosa'][[3.5, 0.2, 'Setosa'],..
#  a partir des listes x, y, variete.
liste = [[x[i], y[i], variete[i]] for i in range(len(x))]

#  Construction d'une liste de la forme [(0.0, 'Setosa'), (0.5, 'Setosa'), (0.2998,
'Setosa'),...
liste_distances = calcule_distances(3.5, 1.2, liste)

#  Construction de la liste precedente triee [(0.0, 'Setosa'), (0.0, 'Setosa'), (0.0,
'Setosa'),...
liste_distances_triee = tri(liste_distances)

#  Construction d'un dictionnaire du type
mon_dico = classe(3, liste_distances_triee)
```

Après l'exécution de ce programme avec les 4 fonctions incomplètes `distance(xA, yA, xB, yB)` `calcule_distances(xP, yP, liste)` `tri(tableau)` et `classe(k,tableauTrie)` on obtient

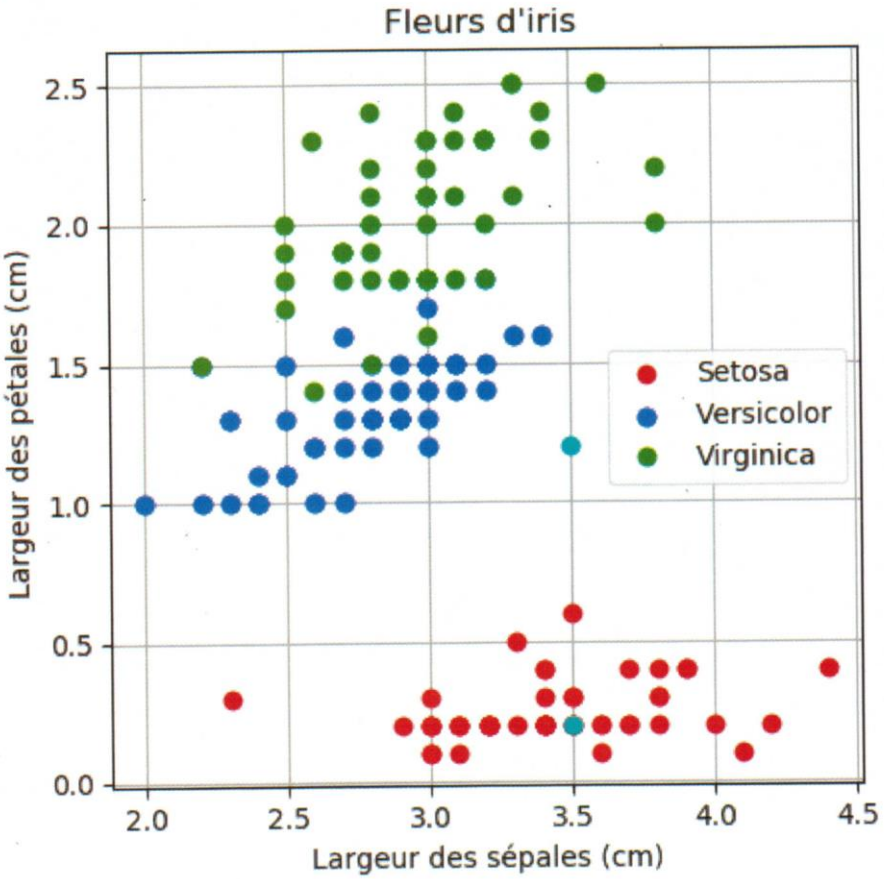
- les valeurs du jeu de données, renvoyées par la fonction `creerListes("irisNum1.csv")` :

```
x =  [3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4, 3.0, 3.0, 4.0,
4.4, 3.9, 3.5, 3.8, 3.8, 3.4, 3.7, 3.6, 3.3, 3.4, 3.0, 3.4, 3.5, 3.4, 3.2, 3.1,
3.4, 4.1, 4.2, 3.1, 3.2, 3.5, 3.6, 3.0, 3.4, ..., 3.4, 3.0]
```

```
y =  [0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.2, 0.1, 0.1, 0.2,
0.4, 0.4, 0.3, 0.3, 0.3, 0.2, 0.4, 0.2, 0.5, 0.2, 0.2, 0.4, 0.2, 0.2, 0.2, 0.2,
0.4, 0.1, 0.2, 0.2, 0.2, 0.2, 0.1, 0.2, 0.2, ..., 2.3, 1.8]
```

```
variete =  ['Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
'Setosa', 'Setosa', 'Setosa', ..., 'Virginica', 'Virginica']
```

- le graphique renvoyé par la fonction `affichePoints()` :



1) Écrire, sous leur *docstring*, le code des quatre fonctions dans les encadrés des pages 7, 8, 9 et 10.

2) On lance le programme de la page 11.

a) Quelle est la valeur de `mon_dico` après l'exécution de ce programme ?

D'après le graphique, le spécimen d'iris correspondant à $xP = 3,5$ et $yP = 1,2$ est proche de 3 iris du jeu de données étiquetés 'Versicolor'. Donc `mon_dico` vaut

{'Setosa': 0, 'Versicolor': 3, 'Virginica': 0}

b) Que peut-on en conclure ?

On peut en conclure que le spécimen est de la classe 'Versicolor'.

c) La valeur de k utilisée dans le programme vous semble-t-elle pertinente ? Sinon, quelle valeur de k proposeriez-vous ?

Le jeu de données contient $n = 150$ iris.

On $\sqrt{150} \approx 12$. Donc $k = 3$ est trop petit. Il faut 12.

Mais comme k doit être impair, on choisit $k = 13$