Exercice 1 : Calculer une moyenne

On veut écrire une fonction qui permet de calculer la moyenne de notes en tenant compte de leurs coefficients.

On propose le code suivant :

```
def moyenne_ponderee(lst_notes, lst_coefs):
    """ lst_notes et lst_coefs sont deux listes de nombres
    Renvoie la moyenne pondérée des notes (float)
    """
    somme_pond = 0
    somme_coefs = 0
    # LA
    for i in range(len(lst_notes)):
        somme_pond = somme_pond + \
                     lst_notes[i] * lst_coefs[i]
        somme_coefs = somme_coefs + lst_coefs[i]
        # ICI
    return somme_pond / somme_coefs
```

a. On utilise la fonction ainsi :

```
notes = [12, 5, 9, 23]
coefficients = [3, 2, 5, 1]
print(moyenne_ponderee(notes, coefficients))
```

Compléter le tableau avec les valeurs des expressions à chaque passage par la ligne indiquée (LA et ICI).

.	i	lst_notes[i]	lst_coefs[i]	somme_pond	somme_coefs
#LA	X				
#ICI	0				
#ICI	1				
#ICI	2				
#ICI	3				

b. Que se passe-t-il lors de l'exécution du code suivant ?

```
notes2 = [12, 5, 9, 23]
coefficients2 = [3, 2, 5, 1, 7]
print(moyenne_ponderee(notes2, coefficients2))
```

c. Que se passe-t-il lors de l'exécution du code suivant ?

```
notes3 = [12, 5, 9, 23]
coefficients3 = [3, 2, 5]
print(moyenne_ponderee(notes3, coefficients3))
```

d. On décide de modéliser les données à l'aide d'une liste de tuples `(note, coefficient)`. Proposer le code d'une fonction qui permet de calculer la moyenne pondérée de ces notes.

Exercice 2 : Vérifier qu'une liste est triée

1- Donner le code d'une fonction *verifie_tri* qui prend en paramètre une liste d'entiers et qui renvoie *True* si la liste est correctement triée dans l'ordre croissant et *False* sinon.

2- Proposer des tests pour cette fonction

Exercice 3 : Chercher un nombre d'occurrences

Écrire une fonction `frequence` qui prend en entrée une chaîne de caractères et un caractère et renvoie le nombre de fois que ce caractère apparaît dans la chaîne (sans utiliser la méthode `count` ou la classe `Counter`). Par exemple :

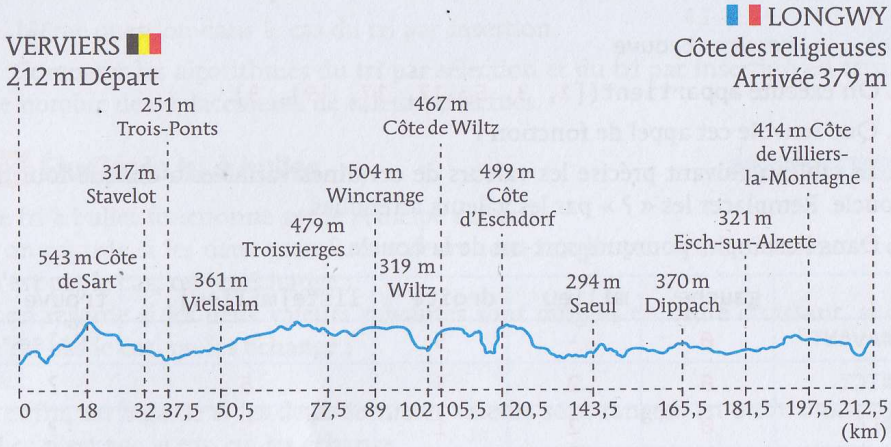
```
>>> frequence("bonjour tout le monde", "o")
4
>>> frequence("salut !!", "o")
0
```

Exercice 4 : Rechercher un extremum

On modélise une étape du Tour de France à l'aide de deux listes : la liste des points de passage et la liste des altitudes de ces points de passage.

Par exemple :

```
points=['Vervier', 'Côte de Sart', 'Stavelot', 'Trois-Ponts',
'Vielsalm', ...]
altitudes=[212, 543, 317, 251, 361, ...]
```



a. Modéliser les données avec deux listes n'est pas une très bonne idée. Proposer une autre modélisation possible.

b. Écrire une fonction qui prend en paramètre les données qui modélisent une étape du Tour de France (structure de données de la question a.) et renvoie le point de passage le plus haut de cette étape.

Pour notre exemple, cette fonction doit renvoyer `'Côte de Sart'`.

c. Écrire une fonction qui prend en paramètre la liste des altitudes d'une étape du Tour de France et renvoie le nombre de descentes.

Avec notre exemple, la liste des altitudes est `[212, 543, 317, 251, 361, 479, 504, 319, 467, 499, 294, 370, 321, 414, 379]` et cette fonction doit renvoyer `6`.

Exercice 5 : Faire une recherche dichotomique dans une liste triée

On propose la fonction suivante :

```
def appartient(liste, element):
    """ indique si element est dans liste
    liste doit être triée dans l'ordre croissant
    """
    gauche = 0
    droite = len(liste) - 1
    trouve = False
    # AVANT
    while gauche <= droite and not trouve:
        milieu = (gauche + droite) // 2
        # ICI
        if liste[milieu] == element:
            trouve = True
        elif liste[milieu] < element:
            gauche = milieu + 1
        else : # liste[milieu] > element
            droite = milieu - 1
        # LA
    return trouve
```

1. On exécute `appartient([1, 3, 5, 17, 17, 19], 3)`.

a. Que renvoie cet appel de fonction ?

b. Le tableau suivant précise les valeurs de certaines variables à chaque tour de boucle. Remplacer les « ? » par les valeurs attendues.

c. Dans l'exemple, pourquoi sort-on de la boucle ?

	gauche	milieu	droite	liste[milieu]	trouve
#AVANT	0	-	5	-	False
#ICI	0	2	5	5	?
#LA	0	2	1	5	?
#ICI	0	0	1	1	?
#LA	1	0	1	1	?
#ICI	1	1	1	3	?
#LA	1	1	1	3	?

2. On exécute `appartient([1, 3, 5, 17, 17, 19], 4)`.

a. Que renvoie cet appel de fonction ?

b. Construire un tableau similaire à celui de la question précédente.

c. Pourquoi sort-on de la boucle ?

3. On exécute `appartient([1, 3, 5, 17, 17, 19], 27)`.

a. Que renvoie cet appel de fonction ?

b. Combien de fois va-t-on passer par la ligne `# ICI` ?

4. En vous inspirant de la fonction `appartient`, écrire le code de la fonction suivante :

```
def indice(liste, element):
    """ renvoie un indice i tel que liste[i] == element
    s'il y en a un, None sinon.
    """
    # À FAIRE

assert indice([1, 3, 5, 17, 17, 19], 3) == 1
assert indice([1, 3, 5, 17, 17, 19], 4) == None
assert indice([1, 3, 5, 17, 17, 19], 17) == 3 or \
       indice([1, 3, 5, 17, 17, 19], 17) == 4
```

Exercice 6 : Tri à bulles

Le tri à bulles fonctionne sur le principe suivant :
- on regarde si les deux premières valeurs sont rangées en ordre croissant, si ce n'est pas le cas, on les échange ;
- on regarde si les deux valeurs suivantes sont rangées en ordre croissant, si ce n'est pas le cas, on les échange ;
- ...
- enfin, on regarde si les deux dernières valeurs sont rangées en ordre croissant, si ce n'est pas le cas, on les échange.

Une fois la première « passe » finie, on recommence. Si la liste contient n valeurs, le tableau sera trié au bout de $n - 1$ passes au plus.

1. Écrire le programme du tri à bulles.

2. À la fin de la première passe, où se trouve la plus grande valeur de la liste ? Dans ces conditions, la seconde passe doit-elle aller jusqu'à comparer les deux dernières valeurs de la liste ? Proposer une version du tri à bulles dans laquelle les passes s'arrêtent de plus en plus tôt dans la liste.

3. Quelle est la complexité du tri à bulles dans la version que vous venez de proposer à la question **2** ?

Exercice 7 : Rechercher le second plus petit élément d'une liste

On souhaite écrire une fonction qui prend en paramètre une liste et renvoie le second plus petit élément de la liste. S'il n'y en a pas, la fonction devra renvoyer la valeur `None`.

Par exemple, si la liste d'entrée contient `[4, 7, 1, -1, 3]`, la fonction devra renvoyer `1`. Si la liste d'entrée est `[4]`, `[]` ou `[3, 3]`, la fonction devra renvoyer `None`.

a. Écrire l'algorithme « naïf » qui consiste à rechercher le plus petit élément, puis à recommencer la recherche en retenant le plus petit élément qui soit différent du plus petit élément déjà trouvé. Attention à la manière d'initialiser la seconde recherche !

b. Étudier la complexité en temps et en nombre de comparaisons de cet algorithme.