Calculatrices et documents non autorisés, à part un aide mémoire recto-verso sur une feuille A4.

Reportez le numéro de votre groupe sur votre copie (-1 point si pas fait). Ce sujet de **4 pages** comprend des exercices indépendants. Au sein d'un même exercice, il est demandé d'utiliser des appels aux fonctions écrites dans les questions précédentes (même si vous n'avez pas donné leur code), lorsque c'est pertinent. Toute fonction doit être réalisée en Python. **Les annotations de type doivent être données**, mais **la documention n'est pas demandée** sauf si la question le précise explicitement. **Le barème est donné à titre indicatif.**

Vous ne devez pas utiliser les fonctions prédéfinies Python `count`, `join`, `del`, `map`, `sum`, `index` et d'une manière générale aucune fonction qui n'aurait pas été vue pendant les enseignements. **Vous devez respecter les bonnes pratiques.**

On rappelle que vous pouvez utiliser l'opérateur `in` qui permet de déterminer si un élément appartient à un itérable et les fonctions `choice` du module `random` de Python et `split` de la classe `str`.

1 Compréhension de code (5.5 points)

1. Pour les trois fonctions suivantes : quand c'est possible, donner un code équivalent en remplaçant la boucle `for` par une boucle `while` ou la boucle `while` par une boucle `for`, en respectant les bonnes pratiques de programmation de l'UE. Quand ce n'est pas possible, écrire "boucle [for/while] équivalente impossible".

```
def bar1(l:list[int]) -> bool:
   for elem in l:
      if elem == 'a':
         return True
   return False
```

```
def bar2() -> list[str]:
   n = 0
   res = []
   while n < 10:
      c = choice('ab')
      res.append(c)
      n = n + 1
   return res
```

```
def bar3() -> list[str]:
   n = 0
   res = []
   while n < 10:
      c = choice('ab')
      res.append(c)
      if c == 'a':
         n = n + 1
   return res
```

2. Donner le contenu du fichier `message.txt` **avant** et **après** l'exécution du code suivant (en partie gauche) :

```
>>> f = open('message.txt', 'r')
>>> f.readline()
'Bonne\n'
>>> f.readlines()
['année\n', 'aux L1 !']
>>> f.close()
>>> with open('message.txt', 'w') as f:
       f.write('Bonne année aux L1 !')
```

```
la = [0, 0]  #1
lb = la      #2
lc = [0, 0]  #3
lc.append(7) #4
lb.append(1) #5
```

3. Pour le code donné ci-dessus en partie droite, donner les **valeurs finales** de `la`, `lb` et `lc` après l'exécution de la dernière instruction et justifier votre réponse en indiquant **l'évolution des valeurs des variables** au cours de l'exécution des instructions 1 à 5 (représenter la mémoire de la manière qui vous semble la plus appropriée).

On considère la fonction mystère suivante :

```
def mystere(liste:list[int], n: int) -> bool:
    """ Précondition : n >= 0 """
    i = 0
    nb = 0
    while i < len(liste) - 1 and nb < n:
        if liste[i] == liste[i+1]:
            nb = nb + 1
        else:
            nb = 0
        i = i + 1
    return nb == n
```

On considère les appels de fonction suivants :

```
I. mystere([1, 1, 1, 2], 2)
II. mystere([], 4)
```

4. Pour chacun des appels :

a. recopier et remplir le tableau de la mémoire ci-dessous de tel sorte qu'il reflète l'exécution de l'appel (les valeurs de `liste[i]` et `liste[i+1]` n'ont pas de sens à l'initialisation). **Le nombre de colonnes est donné à titre indicatif** ;

b. donner le nombre d'itérations effectuées ;

c. expliquer ce qui a causé l'arrêt de la boucle le cas échéant;

d. donner le résultat de l'appel en justifiant à partir de la valeur de `nb`.

`i`							
`liste[i]`							
`liste[i+1]`							
`nb`							

2 Autour des grilles (7 points)

On s'intéresse à des grilles rectangulaires contenant des entiers compris entre 0 et 9 et qui sont délimités par des zones. On aura par exemple la grille ci-dessous (à gauche) à 3 lignes, 2 colonnes et 3 zones, telle que les limites entre les zones apparaissent en gras. On souhaite produire pour la grille une représentation textuelle (au milieu). Les zones sont numérotées de manière arbitraire (à droite).

0	4
2	3
0	1

```
+---+---+
| 0 | 4 |
+---+   +
| 2   3 |
+   +---+
| 0 | 1 |
+---+---+
```

Zone 1	Zone 2
Zone 2	Zone 2
Zone 2	Zone 3

Pour représenter une grille, on utilisera une liste de listes représentant **la liste de ses lignes**, tel que chaque élément de la liste de listes contient un couple (valeur, numéro de zone). On aura pour l'exemple ci-dessus la liste :

```
grille = [[(0, 1), (4, 2)], [(2, 2), (3, 2)], [(0, 2), (1, 3)]]
```

La grille peut être stockée dans un fichier au format texte qui décrit chaque case de la grille et qui contient:

- sur la première ligne : le nombre de lignes n de la grille
- sur la deuxième ligne : le nombre de colonnes m de la grille
- à partir de la troisième ligne : $n * m$ lignes de la forme `<val>,<num_zone>` qui décrivent toutes les cases de la grille de la gauche vers la droite et du haut vers le bas.

On trouve ci-contre le contenu du fichier `grille.txt` correspondant à la grille décrite ci-dessus.

```
3
2
0,1
4,2
2,2
3,2
0,2
1,3
```

5. Écrire une fonction `charger_grille` qui prend en paramètre un nom de fichier de stockage de grille et renvoie une liste de listes de couples d'entiers de type `list[list[tuple[int, int]]]` contenant un couple (valeur, numéro de zone) pour chaque case de la grille. Par exemple `charger_grille('grille.txt')` renvoie la liste `grille` ci-dessus. **Le code doit éviter tout aliasing.**

On cherche à savoir si les éléments d'une zone sont, dans le désordre, les entiers de 1 à la taille de la zone incluse.

6. **Sans utiliser de boucle `while`**, écrire une fonction `est_permutation` qui prend en paramètre une liste non vide d'entiers `liste` de taille `n` et qui renvoie `True` ssi la liste contient (éventuellement dans le désordre) les entiers de 1 à `n`. Par ex `est_permutation([2, 1, 3, 4])` renvoie `True` et `est_permutation([2, 1, 3, 5])` renvoie `False`.

On s'intéresse à la représentation sous forme de chaîne de caractères de la grille, pour affichage. On utilisera les constantes suivantes :

```
SEPH_DIFF = '|'   # séparateur horizontal entre 2 valeurs de zones différentes
SEPH = ' '        # séparateur horizontal entre 2 valeurs de zones identiques
SEPV_DIFF = '---' # séparateur vertical entre 2 zones différentes
SEPV = '   '      # séparateur vertical entre 2 zones identiques
COIN = '+'        # coin d'une case
ESP = ' '         # espacement de part et d'autre d'une valeur
```

et on suppose données[1] les fonctions `zone` et `valeur` qui prennent en paramètre un couple (valeur, numéro de zone):

```
>>> zone((1,2))     >>> valeur((1,2))
2                   1
```

Pour représenter une case on encadre la valeur qu'elle contient par 2 caractères espace.

7. Écrire une fonction `case_formatee` qui prend en paramètre un couple (valeur, numéro de zone) de type `tuple[int, int]` et renvoie une chaîne de caractères qui encadre entre 2 espaces la valeur contenue dans la case passée en paramètre. Par exemple `case_formatee((0,1))` renvoie `ESP + '0' + ESP`.

Pour formater une ligne on sépare chaque représentation de case soit par le caractère `|` (séparateur horizontal `SEPH_DIFF`) si les 2 cases adjacentes n'appartiennent pas à la même zone, soit par le caractère espace (`SEPH`) si les zones sont les mêmes. La chaîne qui représente la ligne commence et termine par le caractère `SEPH_DIFF` (`|`).

8. Écrire une fonction `ligne_formatee` qui prend en paramètre une liste non vide de couples (valeur, numéro de zone), et renvoie sa représentation formatée de type `str`. Par exemple :
   `ligne_formatee([(0, 1), (2, 1), (3, 2)])` vaut `"| 0   2 | 3 |"`.

Une ligne de séparation permet de séparer verticalement 2 lignes de la grille. La représentation utilise un caractère `COIN` (`+`) pour représenter chaque coin de case. Deux cases l'une au dessus de l'autre appartenant à des zones différentes sont séparées par le séparateur vertical `SEPV_DIFF` (`---`), elles sont séparées par un triple espace `SEPV` si leurs zones sont identiques.

9. Écrire une fonction `separation_ligne` qui prend en paramètre 2 lignes de grille non vides de type `list[tuple[int, int]]` et qui renvoie une ligne de séparation de type `str`. Par exemple :
   `separation_ligne([(0, 1), (2, 2), (3, 3)], [(0, 2), (2, 2), (3, 4)])` vaut `"+---+   +---+"` et
   `separation_ligne([(0, 1), (2, 2), (3, 3)], [(1, 1), (2, 2), (0, 3)])` vaut `"+   +   +   +"`.

3 Manipulations de listes (8 points)

Dans cet exercice on considère des listes de caractères. Par exemple :

```
liste1 = ['d', '#', 'y', 'r', 'z', 'g']
liste2 = ['d', '9', '@', 'y', 'z']
liste3 = ['#', 'r', 'g']
```

10. On souhaite écrire une fonction `position_caractere` qui prend en paramètre un caractère et une liste de caractères, et renvoie la position de ce caractère dans la liste, ou la longueur de la liste si le caractère n'est pas dedans. Par exemple, `position_caractere('g', liste1)` vaut 5.

a. Donner des tests pour `position_caractere`;
b. **Sans utiliser de boucle `for`**, écrire la fonction `position_caractere`.

11. Écrire une fonction `au_moins_n_motifs` qui prend en paramètre un entier `n` positif strictement, une liste de caractères `liste` et une liste de caractères non vide `motif` et renvoie `True` ssi la séquence des éléments de `motif` apparaît (dans le même ordre et sans interruption) au moins `n` fois dans `liste`. Par exemple :
    `au_moins_n_motifs(2, ['a', 'b', 'c', 'a', 'b', 'b', 'a'], ['a', 'b'])` vaut `True`
    `au_moins_n_motifs(2, ['a', 'a', 'a'], ['a', 'a'])` vaut `True`
    `au_moins_n_motifs(3, ['a', 'b', 'c', 'a', 'b', 'b', 'a', 'c', 'b'], ['a', 'b'])` vaut `False`.

[1] Vous ne devez PAS coder ces fonctions ni les documenter mais simplement les utiliser en les appelant dans le code de vos fonctions.

12. On souhaite écrire une fonction `genere_liste_sans_begaiement` qui prend en paramètre un entier `n` strictement positif et une liste de caractères `liste` contenant au moins 2 caractères différents, et qui renvoie une liste de caractères de taille `n` construite aléatoirement à partir des éléments de `liste`, telle que 2 caractères qui se suivent ne peuvent pas être identiques (= pas de bégaiement). Par exemple on pourra obtenir pour `genere_liste_sans_begaiement(3, ['s', 'e'])` la liste `['e', 's', 'e']` mais pas la liste `['s', 's', 'e']` (2 caractères `'s'` à la suite).

    a. Donner la précondition de la fonction;
    b. Donner le code de la fonction.

La fonction suivante peut être réalisée même si la fonction qu'elle appelle n'a pas été codée.

13. Écrire une procédure `affiche_sans_begaiement` qui prend en paramètre une liste de caractères `liste` contenant au moins 2 caractères différents. Cette procédure génère à partir de `liste` une liste de 5 caractères sans bégaiement (2 caractères qui se suivent ne peuvent pas être identiques), puis l'affiche.

On aura par exemple pour `affiche_sans_begaiement(['a', 'b', 'c'])` l'affichage :

```
Génération de ['c', 'b', 'c', 'b', 'a'] à partir de ['a', 'b', 'c']
```

14. Écrire un prédicat `est_cache_dans` qui prend en paramètre deux listes de caractères `liste1` (non vide) et `liste2` (quelconque), et qui renvoie `True` ssi tous les caractères de `liste1` apparaissent dans le même ordre dans `liste2`, qui peut contenir éventuellement d'autres caractères placés n'importe où. Par exemple, `est_cache_dans(liste3, liste1)` vaut `True` et `est_cache_dans(['#', 'g', 'r'], liste1)` vaut `False`.