Calculatrices et documents non autorisés.

Reportez le numéro de votre groupe sur votre copie (-1 point si pas fait). Ce sujet de **3 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`, `zip`, `sort`, `map`, `sum`, `index` et d'une manière générale aucune fonction, instruction ou construction qui n'aurait pas été vue pendant les enseignements. **Vous devez respecter les bonnes pratiques. Quand un prédicat utilise une boucle, celle-ci doit être interrompue dès que le résultat de la fonction est connu.**

1 Compréhension de code (4 pts)

1. Pour les deux 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". Ce n'est pas utile de se rappeler exactement ce que réalise la méthode `isupper`.

```
def fun1(s:str) -> bool:
    for c in s:
        if not c.isupper():
            return False
    return True
```

```
def fun2(nb:int) -> str:
    res = ''
    while len(res) < nb:
        c = input()
        res = res + c
    return res
```

2. Pour le code donné ci-dessous, donner les **valeurs finales** de `l1` **et** `l2` **et** `l3` 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).

```
l1 = [1]       #1
l2 = l1 + [2]  #2
l3 = l1        #3
l1[0] = 3      #4
l2[1] = 4      #5
```

On considère la fonction mystère suivante :

```
def mystere(liste:list[int], n:int) -> bool :
    nb = 0                                  #1
    i = 0                                   #2
    while i < len(liste) and nb <= n:       #3
        if liste[i] == 0:                   #4
            nb = nb + 1                     #5
        i = i + 1                           #6
    return nb == n                          #7
```

On considère les appels de fonction suivants :

```
I. mystere([0, 1, 0, 1, 1], 1)
II. mystere([], 0)
```

3. Pour chacun des appels :

a. donner la valeur de `liste`, de `len(liste)` et de `n` ;
b. dans le cas où au moins une itération est effectuée, donner un tableau de la mémoire faisant apparaître les valeurs de `i` et `nb` de telle sorte qu'il reflète l'exécution de l'appel ;
c. donner le résultat de l'appel en justifiant votre réponse par la valeur de `nb`.

2 Autour d'une grille de jeu (7 pts)

Pour cet exercice nous rappelons l'existence des fonctions `copy`, `deepcopy`, `split`, `strip`, `rstrip`, `open`, `readline`, `readlines`, `write`, `writelines` (sans obligation de les utiliser).

On s'intéresse à un jeu de type casse-tête qui se joue sur une grille carrée de taille variable (5, 9, 15…). Une case peut contenir soit un arbre, soit une tente, soit de l'herbe, soit être vide. Initialement chaque case de la grille soit est vide, soit contient un arbre. Le but du jeu est de remplir certaines cases avec soit une tente, soit de l'herbe, les autres cases pouvant rester vides.

On utilisera les constantes globales suivantes :

```
VIDE = ' '
ARBRE = 'A'
TENTE = 'T'
HERBE = 'H'
```

4. Écrire une fonction `saisir_case` qui demande la saisie au clavier d'un caractère qui est soit `HERBE`, soit `TENTE`, fait une nouvelle demande tant que la saisie est incorrecte et renvoie la valeur saisie.

On représentera la grille par une liste de listes de (chaînes de) caractères, chaque liste représentant une **ligne** de la grille.

5. Écrire une fonction `init_grille_vide` qui prend en paramètre un entier strictement positif `taille` et renvoie une liste de listes de caractères qui représente une grille carrée de taille `taille` contenant uniquement des cases valant `VIDE`. **Le code doit éviter tout aliasing.**

La configuration du jeu inclut pour chaque ligne et chaque colonne le nombre exact de tentes à positionner sur cette ligne/colonne (on ne s'intéressera pas plus au détail des règles du jeu). Dans l'affichage chaque ligne/colonne est précédée d'un en-tête de ligne/colonne, un entier positif ou nul qui indique ce nombre de tentes. On donne comme exemple la configuration initiale de jeu ci-dessous avec à gauche l'affichage du jeu et au milieu la grille initiale.

```
    1   0   1
  +---+---+---+
1 |   |   |   |
  +---+---+---+
1 | A | A |   |
  +---+---+---+
0 |   |   |   |
  +---+---+---+
```

```
[[VIDE, VIDE, VIDE], [ARBRE, ARBRE, VIDE], [VIDE, VIDE, VIDE]]
```

Les en-têtes de colonnes sont `1`, `0` et `1`. Donc :

- Les colonnes d'indices `0` et `2` doivent contenir exactement 1 tente.
- La colonne d'indice `1` doit contenir exactement 0 tente.

Les en-têtes de ligne sont `1`, `1` et `0`.

```
3
1 0 1
1 1 0
1 0
1 1
```

6. Écrire une fonction `positionne_colonnes_herbe` qui prend en paramètre une liste `nb_tentes` contenant les en-têtes de colonnes et une grille de même taille que `nb_tentes`. Cette fonction ne modifie pas son paramètre `grille` : elle renvoie **une nouvelle grille**. Pour chaque colonne de la grille qui doit contenir 0 tente d'après `nb_tentes`, la fonction positionne chaque case vide à la valeur `HERBE`. Par exemple pour la liste `[1, 0, 1]` et la grille de l'exemple ci-dessus la fonction renvoie la grille `[[VIDE, HERBE, VIDE], [ARBRE, ARBRE, VIDE], [VIDE, HERBE, VIDE]]`.

On construit l'état initial du jeu à partir d'un fichier de configuration (voir exemple ci-dessus, à droite) qui utilise comme séparateur le caractère espace et qui contient :

- sur la première ligne la taille de la grille ;
- sur la deuxième ligne les en-têtes de colonnes ;
- sur la troisième ligne les en-têtes de lignes ;
- sur chacune des lignes suivantes : les indices de ligne et colonne d'une case contenant un arbre (NB : dans le cas d'une grande grille un indice n'est pas nécessairement un seul chiffre).

On suppose donnée une fonction `liste_nb_tentes` qui prend en paramètre une chaîne de caractères contenant des nombres de tentes séparés par des espaces et renvoie la liste associée. Par exemple `liste_nb_tentes('1 0 1 2')` vaut `[1, 0, 1, 2]`. **Vous ne devez PAS coder cette fonction ni la documenter mais simplement l'utiliser en l'appelant dans le code de vos fonctions.**

Pour renvoyer un triplet contenant les 3 valeurs des variables `x`, `y` et `z` il suffit d'écrire : `return x, y, z`.

7. Écrire une fonction `creer_jeu` qui prend en paramètre le nom d'un fichier contenant la configuration d'un jeu et qui renvoie un triplet contenant la liste des nombres de tentes par colonne (en-têtes de colonnes), la liste des nombres de tentes par ligne (en-têtes de lignes) et la grille contenant les arbres correctement positionnés. **Il n'est pas demandé d'écrire le type renvoyé par la fonction dans sa signature.** Pour l'exemple ci-dessus la fonction renvoie le triplet `([1, 0, 1], [1, 1, 0], [[VIDE, VIDE, VIDE], [ARBRE, ARBRE, VIDE], [VIDE, VIDE, VIDE]])`.

3 Températures et vagues de chaleur (9 pts)

On s'intéresse à des stations météorologiques qui mesurent la température en °C. Pour simplifier nous supposons que la température est un entier et qu'une seule mesure est prise par jour. On représente une série de températures par une liste. Dans cette liste l'élément d'indice `i` représente la température mesurée le jour numéro `i`.

Deux stations sont considérées comme indépendantes si et seulement si les températures mesurées aux mêmes jours sont systématiquement distantes d'au moins 3°C.

8. Le prédicat `sont_stations_independantes` prend en paramètre deux listes d'entiers de même taille représentant des températures et renvoie `True` si la valeur absolue de la différence des températures mesurées aux mêmes jours est toujours supérieure ou égale à 3°C et `False` sinon. Par exemple la fonction renvoie `True` pour les listes `[26, 36, 20]` et `[29, 29, 24]` et `False` pour les listes `[26, 36, 20]` et `[29, 29, 22]` (le dernier couple de mesures ne respecte pas la condition).

a. Donner la précondition de la fonction ;
b. **Sans utiliser de boucle `while`**, écrire le prédicat.

On suppose qu'on dispose pour chaque station de la liste des températures mesurées par ordre croissant.

9. Écrire une fonction `integre_temperatures` qui prend en paramètre 2 listes d'entiers représentant des températures triées par ordre croissant, et renvoie une liste d'entiers contenant les températures des 2 listes triées par ordre croissant. Par exemple pour les listes `[22, 25]` et `[22, 24, 24, 30, 31]`, `integre_temperatures` renvoie `[22, 22, 24, 24, 25, 30, 31]`.

On dit qu'une vague de chaleur se produit :

- soit quand la température mesurée est supérieure ou égale à 26°C pendant au moins un jour ;
- soit quand la température mesurée est supérieure ou égale à 23°C pendant au moins 3 jours consécutifs.

On suppose définie la liste de températures :

`TEMP = [27, 20, 23, 22, 21, 20, 24, 23, 25]`

Cette liste contient 2 vagues de chaleur : température `27` à l'indice `0` et températures `24`, `23` et `25` à partir de l'indice `6`. La liste `[27]` contient une vague de chaleur (`27` à l'indice `0`).

10. Le prédicat `contient_vague_chaleur` prend en paramètre une liste d'entiers représentant des températures et renvoie `True` si la liste contient au moins une suite de températures qui répond aux critères énoncés ci-dessus et `False` sinon.

    a. Donner 3 tests pertinents pour le cas où la fonction renvoie `False` ;
    b. Écrire ce prédicat.

On suppose qu'une vague de chaleur termine dès que 2 températures consécutives inférieures ou égales à 22°C sont mesurées.

11. Écrire la fonction `indice_fin_vague_chaleur` qui prend en paramètre une liste `l_temp` de températures entières contenant au moins une vague de chaleur et un entier `debut` qui représente l'indice de début d'une vague de chaleur dans `l_temp`. Cette fonction renvoie l'indice de la première mesure inférieure ou égale à 22°C qui elle-même suit une mesure inférieure ou égale à 22°C, ou la taille de `l_temp` si 2 telles mesures n'existent pas. Par exemple : `indice_fin_vague_chaleur([27], 0)` vaut 1. La fonction renvoie pour la liste `TEMP` :

- `4` si `debut` vaut `0` (première vague de chaleur terminée par la valeur `21` à l'indice `4`, les valeurs `20` puis `23` ne terminant pas la vague) ;
- `len(TEMP)`, soit `9`, si `debut` vaut `6` (deuxième vague de chaleur non terminée).