Informatique de Tronc Commun

Devoir surveillé n°2

durée : 2 h

La calculatrice n'est pas autorisée

Toutes les réponses sont à donner sur la copie.

Exercice 1. Recherche séquentielle dans une liste

1. Écrire une fonction, `rech`, prenant en argument une liste L et une valeur x, et renvoyant un booléen, `True` ou `False`, selon que x appartient à L ou non. L'utilisation de l'opérateur `in`, pour tester l'appartenance de x à la liste L, est bien sûr interdite dans cet exercice.

- exemples d'appel à la fonction :

```
>>> rech([5, 6, 7], 7)
True
```

```
>>> rech(['f', 'ba', '', 'f'], 'a')
False
```

2. **a.** L'implémentation donnée en réponse à la question **1.** utilise-t-elle un parcours par les indices de L ou un parcours par les valeurs ?

b. Donner une seconde version, `rech2` de la fonction `rech`, utilisant l'autre de ces deux parcours de liste.

3. **a.** Donner un exemple de couple (L, x) où L est une liste de quatre entiers et x un entier, pour lequel le nombre de comparaisons est minimal et un exemple de couple (L, x) où L est une liste de quatre entiers et x un entier, pour lequel le nombre de comparaisons est maximal.

b. De façon générale, si la liste L comporte n éléments, combien de comparaisons (avec l'opérateur « `==` ») sont-elles effectuées, au minimum ? au maximum ?

On suppose maintenant que la liste L dans laquelle on effectue la recherche d'une valeur x, est une liste d'entiers **triée dans l'ordre croissant.**

4. **a.** Si l'on fait appel à l'une des deux fonctions précédentes, `rech` ou `rech2`, combien de comparaisons de x avec des éléments de L sont-elles nécessaires pour examiner si la valeur $x = 0$ appartient à la liste L égale à $[1, 2, 3, 4]$?

b. Si l'on tient compte du fait que la liste L est triée, expliquer pourquoi une seule comparaison pourrait suffire pour déterminer si la valeur $x = 0$ appartient à la liste $L = [1, 2, 3, 4]$?

c. Proposer une version `rech3` de la fonction de recherche, qui, **lorsque la liste en entrée est triée dans l'ordre croissant**, permet de minimiser le nombre de comparaisons de x avec des éléments de L, en tirant parti de ce que la liste est triée.

Exercice 2. Recherche de la position du maximum d'une liste d'entiers

On donne la fonction suivante, prenant en argument des listes d'entiers :

```
1.  def f(L):
2.      imax, vmax = None, 0
3.      for i in range(len(L)):
4.      if L[i] >= vmax:
5.          vmax = L[i]
6.          imax = i
7.      return imax, vmax
```

1. **a.** Que renvoie l'appel `f([5, 4, 3, 5])` ?

b. Que renvoie l'appel `f([-5, -4, -3, -5, -3])` ?

On souhaite que la fonction `f` renvoie la position de la première occurrence du maximum de la liste L en entrée.

2. Comment modifier le code de la fonction `f` pour qu'il en soit ainsi ?

On souhaite maintenant modifier le code de la fonction `f` pour écrire une fonction `g` qui renvoie les deux plus grandes valeurs de la liste d'entiers *L* en entrée, pour une liste *L* non nécessairement triée, de longueur supérieure ou égale à 2.

On obtiendrait ainsi, par exemple :

```
>>> g([4, 5, 2, 3, 2, 7])
(5, 7
```

```
>>> g([1, -5, 2, -3, 2, -7])
(2, 2)
```

3. Proposer une implémentation pour la fonction `g`.

Exercice 3. Fonctions mystères opérant sur des listes

1. On considère les fonctions `mystere1` et `mystere2` suivantes prenant en entrée des listes d'entiers de longueur au moins 2 :

```
1. def mystere1(L):
2.     for i in range(len(L) - 1):
3.         if L[i] == L[i + 1]:
4.             return False
5.         return True
```

```
1. def mystere1(L):
2.     for i in range(len(L) - 1):
3.         if L[i] == L[i + 1]:
4.             return False
5.     return True
```

a. Que renvoie les appels `mystere1([4, 5, 6, 3, 7, 6])` ? `mystere1([4, 5, 6, 6, 3, 7])` ?

b. Que renvoie les appels `mystere2([4, 5, 6, 3, 7, 6])` ? `mystere2([4, 5, 6, 6, 3, 7])` ?

c. Indiquer ce que fait chacune de ces deux fonctions. (Il s'agira de préciser ce que renvoie chacune des deux fonctions en fonction des caractéristiques de la liste en entrée et non de paraphraser le code de la fonction).

2. On considère la fonction `mystere3` suivante, prenant en argument des listes d'entiers :

```
1. def mystere3(L):
2.     A = []
3.     for i in range(1, len(L)):
4.         if L[i - 1] <= L[i] and L[i] >= L[i + 1]:
5.             A.append(L[i])
6.     return A
```

a. L'appel `mystere3([1, 2])` déclenche une erreur. Laquelle et pourquoi ?

b. Caractériser les listes en entrée qui déclenchent la même erreur qu'en **a.** et celles qui ne la déclenchent pas.

c. Proposer une modification minimale du code de la fonction `mystere3`, de sorte qu'aucun appel sur des listes d'entiers ne déclenche l'erreur précédente (recopier uniquement la(les) ligne(s) à modifier).

Dans les deux questions suivantes, on suppose que l'erreur précédente a été corrigée.

d. Que renvoie les appels `mystere3([4, 5, 6, 3, 7])` ? `mystere3([4, 5, 6, 3, 7, 6])` ?

e. Que renvoie les appels `mystere3([])` ? `mystere3([8, 4, 5, 6, 1, 3, 3, 7, 6, 3, 4, 2])` ?

f. Que fait la fonction `mystere3` ? (on caractérisera la liste en sortie en fonction de la liste en entrée).

3. On considère la fonction `mystere4` suivante, prenant en argument une liste d'entiers :

```
1. def mystere4(L):
2.     A = []
3.     i = 0
4.     while i < len(L):
5.         A.append(L[i])
6.         i = i + 2
7.     return A
```

a. Que renvoient les appels

- `mystere4([5, 8, 2, 3])` et
- `mystere4([5, 8, 2, 3, 4])` ?

b. De façon générale, que renvoie la fonction `mystere4` ?

Exercice 4. Compréhensions de listes

1. Définir une fonction `f(L)` renvoyant une copie de la liste *L* en entrée obtenue à l'aide d'une compréhension de liste.

2. Définir une fonction `g(n)` renvoyant la liste des carrés des entiers de l'intervalle $[\![0, n]\!]$ obtenue à l'aide d'une compréhension de liste.

Exercice 5. Chaînes de caractères

On rappelle le principe de construction d'une chaîne caractère par caractère à partir d'une chaîne vide au travers de la donnée de la fonction suivante, `copie`, qui permet de construire et de renvoyer une copie d'une chaîne de caractères, `ch`.

```
def copie(ch):
    ch1 = ""
    for i in range(len(ch)):
        ch1 = ch1 + ch[i]
    return ch1
```

Les questions 1, 2 et 3 sont indépendantes entre elles.

1. Écrire une fonction `remplace(ch, old, new)` renvoyant une copie d'une chaîne `ch` en entrée dans laquelle les occurrences du caractère `old` ont été remplacée par le caractère `new`.
   Par exemple, l'appel `remplace("toto", "o", "i")` renverra la chaîne `"titi"`.
2. Écrire une fonction `plpc(ch1, ch2)` renvoyant le plus long préfixe commun aux chaînes `ch1` et `ch2`, c'est-à-dire la plus longue chaîne `ch` telle que `ch` coïncide avec les premiers caractères des chaînes `ch1` et `ch2`.
   On doit obtenir, par exemple :

```
>>> plpc('toto', 'titi')
't'
>>> plpc('toto', 'abcdef')
''
```

```
>>> plpc('carte', 'carton')
'cart'
>>> plpc('cartes', 'carte')
'carte'
```

3. Écrire une fonction `renverse(ch)` renvoyant une copie d'une chaîne `ch` dans laquelle l'ordre des caractères est inversé. Par exemple, l'appel `renverse("abcd")` renverra la chaîne `"dcba"`.
4. **a.** Écrire une fonction `teste(ch)`, **faisant appel à la fonction `renverse`** et renvoyant un booléen, `True` ou `False`, selon que la chaîne en entrée est un palindrome ou non. Par exemple, l'appel `teste("toto")` renverra `False`, et `teste("kayak")` renverra `True`.
   b. L'appel à la fonction `renverse` a le désavantage d'effectuer systématiquement une copie de la chaîne `ch` en entrée, ce qui prend un temps et occupe un espace en mémoire, proportionnels à la longueur de la chaîne `ch`. Proposer une fonction `teste2(ch)` corrigeant cet inconvénient en n'effectuant aucune copie de la chaîne `ch`.

Exercice 6. Occurrences d'un motif dans une chaîne de caractères

On rappelle la syntaxe du *slicing* pour les chaînes de caractères :

Pour une chaîne `ch`, et deux entiers a et b tels que $0 \leq a \leq b <$ len(ch), `ch[a:b]` est une copie de la portion de la chaîne `ch` constituée des caractères de la chaîne `ch` situés aux positions $a, a+1, \ldots, b-1$.

Par exemple, si `ch = "abracadrabra"`, de longueur 11, `ch[4:5]` vaut `"c"`, `ch[8:11]` vaut `"bra"` et `ch[3:3]` est une chaîne vide.

On rappelle que dans ces conditions, la chaîne `ch[a:b]` a pour longueur $b - a$.

On considère la fonction `nombre1`, définie ci-dessous, dont l'implémentation est incomplète :

```
1. def nombre1(motif, texte):
2.     M = len(motif)
3.     T = len(texte)
4.     nb_occ = ...
5.     for i in range(...):
6.         if texte[... : ...] == ... :
7.             ...
8.     return ...
```

La fonction proposée, `nombre1`, une fois complétée correctement, doit renvoyer le nombre d'occurrences de la chaîne `motif` dans la chaîne de caractères `texte`.

On doit obtenir, par exemple :

```
>>> nombre1("les", "les MPSI et les PCSI sont en devoir d'informatique")
2
>>> nombre1("ique", "les MPSI et les PCSI sont en devoir d'informatique")
1
```

1. Recopier et compléter les lignes 4 à 8 du programme précédent.

Pour éviter des copies inutiles de chaînes de caractères, on cherche à modifier la fonction `nombre1` en une fonction `nombre2` en recourant à une implémentation n'utilisant pas le *slicing*.

Une implémentation incomplète de la fonction `nombre2` est proposée ci-dessous :

```
1.  def nombre2(motif,texte):
2.          M = len(motif)
3.          T = len(texte)
4.          nb_occ = ...
5.          for i in range(...):
6.              k = 0
7.              while k < M and texte[...] == motif[...]:
8.                  k = k + 1
9.              if k == ... :
10.                 ...
11.         return ...
```

2. Recopier et compléter les lignes 6 à 10 du programme ci-dessus (les lignes 4, 5 et 11 restant, elles, identiques aux lignes 4, 5 et 8 complétées à la question **1.**).

Exercice 7. Dictionnaires

On rappelle la syntaxe pour définir un dictionnaire vide, `d` :

```
d = {} # ou d = dict()
```

On rappelle la syntaxe pour ajouter un couple $(cle, valeur)$ à un dictionnaire `d` :

```
d[cle] = valeur
```

On rappelle que la syntaxe est la même pour modifier la valeur associée à une clé déjà existante.

On rappelle les trois parcours possibles pour un dictionnaire `d` :

```
## parcours par les clés
for cle in d.keys():
    ...
## parcours par les clés et les valeurs
for cle, val in d.items():
    ...
## parcours par les valeurs
for val in d.values():
    ...
```

On considère les notes obtenues à un contrôle, pour lequel on a dénombré les nombres d'élèves ayant obtenu une note comprise dans chacun des intervalles de notes $[0; 4[$, $[4; 8[$, $[8; 12[$, $[12; 16[$ et $[16; 20]$.

On a enregistré les résultats dans le dictionnaire suivant :

```
d = {2: 5, 6: 7, 10: 12, 14: 9, 18: 3}
```

dont les clés sont les entiers correspondant au milieu de l'intervalle considéré. On considère pour la suite que les notes obtenues sont représentées par le milieu de l'intervalle dans lequel chacune se trouve.

Ainsi, pour ce contrôle, 7 élèves ont obtenu une note comprise entre 4 et 8, strictement inférieure à 8.

1. a. Écrire une fonction `effectif_total` prenant en argument un tel dictionnaire, et renvoyant le nombre total d'élèves évalués. Sur l'exemple la fonction renverra 36.

b. Écrire une fonction `moyenne` prenant en argument un tel dictionnaire, renvoyant la moyenne des notes obtenues, en pondérant chaque note, 2, 6, 10, 14 et 18 par l'effectif correspondant.

2. Écrire une fonction `histogramme`, prenant en argument une liste de notes entières comprises entre 0 et 20 et renvoyant un dictionnaire construit sur le principe précédent.

On pourra utiliser le squelette de fonction suivant et les opérations de division entière et de modulo.

```
1.  def histogramme(L):
2.          d = {2: ..., 6: ..., ...}
3.          for note in L:
4.              # determination de la clé associée à la note
5.              ... # utiliser ici une ou plusieurs lignes
6.              # mise à jour de la valeur associée à la clé
7.              d[...] = ...
8.          return d
```

Pour les exercices **8** et **9**, on rappelle les fonctions et méthodes suivantes utiles pour lire et écrire dans des fichiers textes :

- `f = open(fichier, mode)` crée un descripteur de fichier, `f`, permettant d'obtenir l'accès à un fichier. Le mode peut être `'r'` (lecture), `'w'` (écriture) ;
- `f.close()` ferme l'accès au fichier auquel l'accès a été demandé par `f`.

Si `f` est un descripteur de fichier donnant accès à un fichier ouvert en lecture :

- `f.read()` renvoie l'intégralité du contenu du fichier sous la forme d'une unique chaînes de caractères ;
- `f.readline()` renvoie une chaînes de caractères correspondant aux caractères d'une ligne du fichier, caractère de saut de ligne, `'\n'`, inclus (la lecture commence à la première ligne du fichier, et après chaque appel à la méthode `.readline()`, une lecture suivante dans le fichier commencera à la ligne suivante) ;
- `f.readlines()` renvoie une liste de chaînes de caractères dont chacune correspond aux caractères composant une ligne du fichier, caractère de saut de ligne, `'\n'`, inclus.

Si `f` est un descripteur de fichier donnant accès à un fichier ouvert en écriture :

- `f.write(S)` écrit dans le fichier la chaîne de caractères `S`.

On rappelle les méthodes `.strip()` et `.split()` sur les chaînes de caractères.
`S` désignant une chaîne de caractères :

- `S.strip()` renvoie une nouvelle chaîne de caractères qui est une copie de `S`, privée de son dernier caractère si celui-ci est une espace ou un caractère de passage à la ligne `'\n'` ;
- `S.split(sep)`, où `sep` désigne une chaîne de caractères, renvoie une liste de chaîne de caractères, qui est la liste de toutes les portions de la chaîne `S` (sans en changer l'ordre) qui sont comprises entre le début de la chaîne `S` et la première occurrence de `sep`, ou entre deux occurrences consécutives de la chaîne `sep` dans `S`, ou entre la dernière occurrence de `sep` dans `S` et la fin de la chaîne `S`.

Par exemple, on aura :

```
>>> 'ababca'.split('a')
['', 'b', 'bc', '']
>>> '1,3,4'.split(',')
['1', '3', '4']
```

On rappelle les fonctions de conversion de type :

- `int` qui, appliquée à une chaîne de caractères représentant un chiffre, renvoie l'entier correspondant ;
- `float` qui, appliquée à une chaîne de caractère représentant un flottant, renvoie la valeur flottante correspondante ;
- `str` qui, appliquée à un entier ou un flottant, renvoie l'écriture décimale du nombre sous la forme d'une chaîne de caractères.

Par exemple :

```
>>> int('23')
23
```

```
>>> float('-4.3')
4.3
```

```
>>> float('3e-2')
0.03
```

```
>>> str(102)
'102'
```

Exercice 8. Écriture dans un fichier

Donner une séquence d'instructions permettant de créer le fichier d'extension **.py** dont le contenu est le suivant, de sorte qu'il soit exécutable en tant que programme Python.
On utilisera le caractère `'\t'` pour encoder les indentations.

prog.py

```
for k in range(10):
    print('bonjour')
```

Exercice 9. Fréquence des caractères dans un texte

On considère un fichier texte, **texte.txt** ne contenant que des caractères pris parmi les 26 lettres minuscules de l'alphabet latin, et des espaces ou des passages à la ligne, et dont on suppose qu'il est présent dans le répertoire de travail courant au moment de l'appel aux fonctions que l'on écrira.
Par exemple :

texte.py

```
joyeux noel a tous les
eleves de pcsi et de mpsi
```

1. Définir une fonction `load(chemin)` qui, si `chemin` est une chaîne de caractères donnant le chemin vers un fichier texte, renvoie le contenu du fichier sous la forme d'une chaîne de caractères.
   Écrire ensuite l'appel à la fonction `load` renvoyant le contenu du fichier donné en exemple.
2. Compléter la fonction `init()` suivante, renvoyant un dictionnaire `d` dont les clés sont les 26 lettres de l'alphabet latin et les deux caractères `' '` (espace) et `'\n'`, associées chacune à l'entier 0.
   On n'écrira sur la copie que les lignes manquantes.

```
1. def init():
2.     alphabet = 'abcdefghijklmnopqrstuvwxyz'
3.     ...
4.     return d
```

On devra avoir :

```
>>> init()
{'a': 0, 'b': 0, 'c': 0, 'd': 0, 'e': 0, 'f': 0, 'g': 0, 'h': 0, 'i': 0, 'j': 0,
'k': 0, 'l': 0, 'm': 0, 'n': 0, 'o': 0, 'p': 0, 'q': 0, 'r': 0, 's': 0, 't': 0, 'u':
0, 'v': 0, 'w': 0, 'x': 0, 'y': 0, 'z': 0, ' ': 0, '\n': 0}
```

3. Définir une fonction `compte_occ(chemin)`, faisant appel aux fonctions `load` et `init`, qui, si `chemin` est une chaîne de caractères donnant le chemin vers un fichier texte, renvoie un dictionnaire ayant la forme du dictionnaire `d`, et dans lequel chaque caractère est associé au nombre d'occurrences de ce caractère dans le fichier.
   L'appel à la fonction `compte_occ` pour le fichier donné en exemple devra renvoyer le dictionnaire suivant :

```
{'a': 1, 'b': 0, 'c': 1, 'd': 2, 'e': 9, 'f': 0, 'g': 0, 'h': 0, 'i': 2, 'j': 1,
'k': 0, 'l': 3, 'm': 1, 'n': 1, 'o': 3, 'p': 2, 'q': 0, 'r': 0, 's': 5, 't': 2, 'u':
2, 'v': 1, 'w': 0, 'x': 1, 'y': 1, 'z': 0, ' ': 9, '\n': 1}
```

Exercice 10. Lecture d'un fichier .csv

Les fichiers **.csv** sont des fichiers texte, permettant d'encoder des tableaux construits à l'aide d'un tableur.
On considère le tableau suivant dans lequel sont enregistrées des coordonnées (entières) de points du plan, tel qu'il se présente dans un tableur :

	A	B	C	D	E
1	point	abscisse	ordonnée		
2	A1	4	5		
3	A2	-1	6		
4	...	...	...		

Enregistré au format **.csv**, ce tableau se présente sous la forme suivante :

coords.csv

```
point;abscisse;ordonnée
A1;4;5
A2;-1;6
...
```

1. Compléter le script suivant afin de recueillir la première ligne du fichier dans une variable `entetes`, de type `str`, et, dans une liste `L`, les lignes suivantes. Le premier élément de `L` sera la chaîne `'A1;4;5\n'`. On pourra utiliser, selon les besoins, les méthodes `.read()`, `.readline()`, ou `.readlines()`.

```
f = open(...)
entetes = ...
...
```

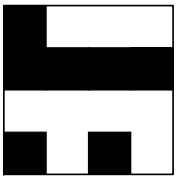
On souhaite écrire une fonction `traitement(ligne, sep)`, prenant en argument une chaîne `ligne` terminant par un caractère de passage à la ligne, `'\n'`, et un caractère `sep` et renvoyant une liste de toutes les sous-chaînes de la chaîne `ch`, ne contenant pas le séparateur sep, et délimitées, à gauche, par le début de la chaîne `ch` ou une occurrence du séparateur `sep`, et, à droite, une occurrence du séparateur `sep` ou la fin de la chaîne `ch` privée d'un éventuel caractère de passage à la ligne.
Par exemple, l'appel `traitement('abracadabra\n', 'a')` renverra la liste `['', 'br', 'c', 'd', 'br', '']`.

2. Écrire une première version de la fonction `traitement`, faisant appel aux méthodes `.strip()` et `.split()`.
3. **On suppose pour cette question que le séparateur ne comporte qu'un seul caractère**. Proposer une seconde version de la fonction `traitement`, ne faisant pas appel à ces deux méthodes, mais utilisant uniquement les opérations de base sur les chaînes de caractères et éventuellement le *slicing*.
4. Utiliser la fonction `traitement` afin de construire, à partir de la liste `L`, la liste de couples d'entiers, `coords`, contenant les coordonnées de tous les points du fichier.
   Sur l'exemple, la liste `coords` sera la liste `[(4, 5), (-1, 6), ...]`.
5. Utiliser la fonction `traitement` afin de construire, à partir de la liste `L`, une liste de dictionnaires, `dcoords`, dont chaque élément est un dictionnaire de la forme `{'point' : ..., 'x' : ..., 'y' : ...}`.
   Sur l'exemple, le premier élément de la liste `dcoords` serait le dictionnaire :
   `{'point' : 'A1', 'x' : 4, 'y' : 5}`.

Exercice 11. Traitement d'images

On considère une liste `L` qui est composée de n listes à n éléments ($n \geq 1$) égaux à 0 ou 1.
La liste de listes `L` est une représentation d'une image en noir et blanc, où 0 représente un pixel noir et 1 un pixel blanc.

- Exemple :
  La liste `L0 = [[0, 1, 1, 1], [0, 0, 0, 0], [1, 1, 1, 1], [0, 1, 0, 1]]` code l'image suivante, à quatre lignes et quatre colonnes :



0	1	1	1
0	0	0	0
1	1	1	1
0	1	0	1

On considère la fonction traitement ci-dessous :

```
1. def traitement(L):
2.       n, p = len(L), len(L[0])
3.       for i in range(n // 2):
4.           for j in range(p):
5.               L[i][j], L[n - 1 - i][j] = L[n - 1 - i][j], L[i][j]
```

On appelle la fonction traitement avec pour paramètre la liste `L0` définie en exemple.

1. Lors de l'exécution de l'appel `traitement(L0)`, quelles sont les valeurs prises par les variables `n` et `p` ? Quelles valeurs prennent ces variables pour une image rectangulaire quelconque ?
2. À l'issue de l'exécution de l'appel `traitement(L0)`, quel est le contenu de la liste `L0` ? Quelle image est alors codée par `L0` ? On dessinera cette image sur la copie.
   Décrire en une phrase l'effet de l'application `traitement` sur l'image codée.
3. Proposer une implémentation alternative, `traitement1`, de la fonction `traitement` n'utilisant qu'une boucle `for` (et aucune de boucle `while`).

On demande maintenant, d'écrire plusieurs versions modifiées de la fonction `traitement`, qui modifient la liste de liste en entrée, en produisant des effets spécifiques sur l'image codée.

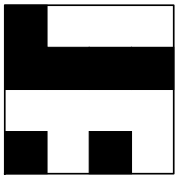
4. Modifier la fonction `traitement` en une fonction `negatif`, transformant la liste de liste en entrée en la liste de liste codant le négatif de l'image codée par la liste en entrée. Cette fonction agira sur la liste de liste en entrée en y remplaçant les 1 par des 0 et les 0 par des 1.
Ainsi, par l'appel `negatif(L0)`, la liste

```
L0=[[0, 1, 1, 1], [0, 0, 0, 0], [1, 1, 1, 1], [0, 1, 0, 1]]
```

sera modifiée en la liste

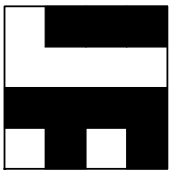
```
[[1, 0, 0, 0], [1, 1, 1, 1], [0, 0, 0, 0], [1, 0, 1, 0]]
```

et l'image



0	1	1	1
0	0	0	0
1	1	1	1
0	1	0	1

devient



1	0	0	0
1	1	1	1
0	0	0	0
1	0	1	0

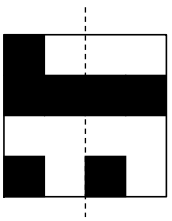
5. Modifier la fonction traitement en une fonction `symetrie_verticale`, qui agit sur la liste de liste en entrée de sorte que l'image codée soit transformée par une symétrie axiale par rapport à la médiatrice des côtés haut et bas de l'image (en pointillés ci-dessous).
Ainsi, par l'appel `symetrie_verticale(L0)`, la liste

```
L0 =[[0, 1, 1, 1], [0, 0, 0, 0], [1, 1, 1, 1], [0, 1, 0, 1]]
```

sera modifiée en la liste

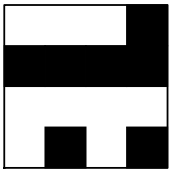
```
[[1, 1, 1, 0], [0, 0, 0, 0], [1, 1, 1, 1], [1, 0, 1, 0]]
```

et l'image



0	1	1	1
0	0	0	0
1	1	1	1
0	1	0	1

devient



1	1	1	0
0	0	0	0
1	1	1	1
1	0	1	0

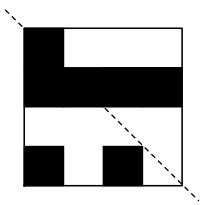
6. Modifier la fonction traitement en une fonction `symetrie_diagonale`, qui agit sur la liste de liste en entrée de sorte que l'image codée soit transformée par une symétrie axiale par rapport à la diagonale « descendante » (en pointillés ci-dessous) de l'image, **supposée carrée pour cette question**.
Ainsi, par l'appel `symetrie_diagonale(L0)`, la liste

```
L0 =[[0, 1, 1, 1], [0, 0, 0, 0], [1, 1, 1, 1], [0, 1, 0, 1]]
```

sera modifiée en la liste

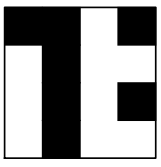
```
[[0, 0, 1, 0], [1, 0, 1, 1], [1, 0, 1, 0], [1, 0, 1, 1]]
```

et l'image



0	1	1	1
0	0	0	0
1	1	1	1
0	1	0	1

devient



0	0	1	0
1	0	1	1
1	0	1	0
1	0	1	1