

Informatique Tronc Commun
Devoir surveillé n°1

Lycée Fabert de Metz, MPSI – PCSI (2025 – 2026)

A compléter

NOM :

Prénom :

Classe :

Cadre réservé au
correcteur

Note : / 20

Durée de l'épreuve : 1h00

Consignes :

- La correction tient compte du soin apporté à la **présentation** et à la **rédaction** de la copie. Les bouts de code devront être écrits en **langage Python** en respectant la syntaxe inhérente à ce langage. On veillera notamment à faire apparaître clairement les **indentations** qui structurent le code.
- Lorsque l'énoncé impose des contraintes sur le type et la valeur des arguments d'une fonction, on supposera que ces contraintes sont effectivement vérifiées lors de l'appel de la fonction. Il n'est donc pas nécessaire d'ajouter des tests dans la fonction pour vérifier ces contraintes.

Exemple : si l'énoncé demande de définir une fonction $f(x)$ avec x un entier positif, il n'est pas nécessaire dans la définition de la fonction f de contrôler le type et le signe de x .

- L'usage des **calculatrices** et de **tout autre document** n'est **pas autorisé**.

Les questions abordées dans ce problème ne sont pas toutes indépendantes : pour y répondre, on pourra faire appel aux fonctions des questions précédentes, même si celles-ci n'ont pas été traitées.

Problématique

Supposons que l'on dispose d'une liste L dont tous les éléments sont des entiers (la taille de la liste peut éventuellement être très grande). On souhaite savoir s'il est possible d'obtenir une valeur v donnée en sommant deux entiers contenus dans la liste L (sans possibilité de sommer un élément avec lui-même).

Par exemple, en considérant une liste $L = [5, 0, 1, 6, 9, 3, 4]$, on constate :

- qu'il est possible de générer la valeur $v = 7$ en sommant les entiers 1 et 6 (on pourrait également sommer les entiers 3 et 4)
- qu'il est impossible de générer les valeurs $v = 2$ ou $v = 16$ en sommant une paire d'entiers présents dans la liste.

On propose par la suite différentes stratégies pour résoudre ce type de problème.

A. Recherche aléatoire

On se propose dans un premier temps de miser sur la chance pour identifier une paire d'entiers dont la somme possède effectivement la valeur ciblée. On suppose que les variables L (liste d'entiers) et v (valeur ciblée) ont déjà été créées et on dispose également d'une fonction `alea(i_min, i_max)` :

- qui prend en arguments deux entiers $i_{\min}$ et $i_{\max}$
- qui renvoie un couple (i_1, i_2) où les entiers i_1 et i_2 sont choisis aléatoirement, en vérifiant $i_1 \in \llbracket i_{\min} ; i_{\max} \rrbracket$, $i_2 \in \llbracket i_{\min} ; i_{\max} \rrbracket$ et $i_1 \neq i_2$.

Exemple d'application :

```
In : alea(3, 10)
Out: (5, 4)
In : alea(3, 10)
Out: (7, 10)
In : alea(3, 10)
Out: (3, 8)
```

Pour les questions associées à cette partie A, on ne demande pas d'écrire des « fonctions ».

A.1. En exploitant cette fonction `alea`, écrire les instructions permettant de créer deux entiers `i1` et `i2` correspondant à un couple (i_1, i_2) d'indices de position distincts et valides pour la liste L . On commencera pour cela par créer une variable `n` correspondant à la taille de la liste L .

/ 1,5 pt

A.2. L'instruction précédente ayant été exécutée, écrire les instructions permettant de créer une variable booléenne nommée `reponse` :

- qui vaut `True` si la somme des entiers situés à ces indices de position `i1` et `i2` vaut effectivement `v`
- qui vaut `False` sinon.

/ 1 pt

A.3. Ecrire les instructions permettant de répéter le tirage aléatoire des indices `i1` et `i2` tant que la somme des entiers correspondants est différente de la valeur `v` ciblée, puis affichant dans la console le couple (i_1, i_2) approprié (en supposant ici qu'un tel couple existe). L'initialisation des variables `i1` et `i2` est attendue ici.

/ 2 pt

A.4. Comme cet algorithme de recherche aléatoire ne se terminera pas forcément, adapter les instructions (les réécrire ci-dessous) pour que les tirages aléatoires s'interrompent après 1000 itérations dans le pire des cas (si la valeur cible v n'a toujours pas été atteinte). On affichera finalement dans la console une valeur booléenne valant respectivement `True` ou `False` si la recherche a abouti ou non.

/ 2 pt

B. Recherche exhaustive

Une autre stratégie de résolution consiste à tester toutes les combinaisons possibles de paires d'entiers. Comme la somme est une opération commutative, on pourra en pratique se contenter d'énumérer les couples (i_1, i_2) d'indices de position tels que $i_2 < i_1$.

B.1. On représente ci-contre une grille permettant de lister tous les couples (i_1, i_2) d'indices de position pour une liste L possédant six éléments. Mettre une croix dans toutes les cases respectant la condition attendue $i_2 < i_1$. Le résultat obtenu aidera à la résolution de la question suivante.

B.2. Pour énumérer tous les couples d'indices nécessaires, on peut ainsi s'appuyer sur une structure de deux boucles imbriquées portant sur les indices `i1` et `i2`. Pour une liste L donnée, compléter en conséquence le bout de code suivant (compléter les lignes d'instructions numérotées 1, 2 et 3).

/ 1 pt

		indice i_2					
		0	1	2	3	4	5
indice i_1	0						
	1						
	2						
	3						
	4						
	5						

```

1  n =                # Taille de la liste L
2  for i1 in
3      for i2 in

```

/ 1,5 pt

B.3. En s'appuyant sur le principe de cette recherche exhaustive, écrire une fonction `rechercheB(L, v)` :

- qui prend en arguments une liste d'entiers L et un entier v (valeur cible)
- qui renvoie un tuple $(i1, i2)$ tel que la somme des entiers de positions `i1` et `i2` est égale à v si une telle paire d'entiers existe, et qui renvoie `False` sinon.

Exemple d'application :

```

In : L = [5, 0, 1, 6, 9, 3, 4]
In : rechercheB(L, 7)
Out: (3, 2)
In : rechercheB(L, 2)
Out: False

```

C. Filtrage de la liste

Pour une valeur cible v donnée, il est en pratique possible de réduire la taille de la liste des entiers à considérer si l'on connaît la valeur minimale $e_{\min}$ des entiers e contenus dans la liste L initiale. En effet, tous les entiers e vérifiant $e + e_{\min} > v$ ne pourront assurément pas engendrer la valeur cible v par sommation avec un autre entier de la liste L . Cela permettra donc de poursuivre la résolution avec une liste de taille réduite, ce qui accélère la recherche.

C.1. Ecrire une fonction `minimum(L)` :

- qui prend en argument une liste d'entiers L
- qui renvoie la valeur minimale parmi tous les entiers contenus dans la liste.

Exemple d'application :

```
In : L = [5, 8, 6, 2, 9, 3, 4]
In : minimum(L)
Out: 2
```

C.2. Ecrire une fonction `filtrage(L, v)` :

- qui prend en arguments une liste d'entiers L et un entier v (valeur cible)
- qui crée une variable locale `e_min` correspondant à la valeur minimale contenue dans la liste L
- qui renvoie une nouvelle liste L_{new} ne contenant que les entiers effectivement « utiles » (ceux pouvant potentiellement engendrer la valeur cible v par sommation).

Exemple d'application :

```
In : L = [5, 8, 6, 2, 9, 3, 4]
In : filtrage(L, 7)
Out: [5, 2, 3, 4]
```

D. Algorithme appliqué à une liste triée

On peut envisager un dernier algorithme de recherche de cette paire d'entiers, mais qui nécessite cette fois de se restreindre à une liste L d'entiers triée par valeurs croissantes.

D.1. Ecrire une fonction `triee(L)` :

- qui prend en argument une liste d'entiers L
- qui renvoie `True` si la liste est triée par valeurs croissantes (mais pas forcément « strictement » croissantes) et qui renvoie `False` sinon.

Exemple d'application :

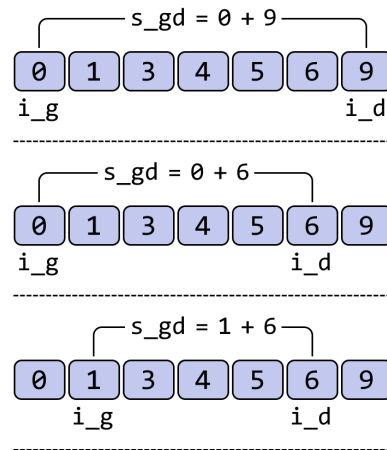
```
In : L1 = [5, 8, 5, 2, 9, 3, 4]
In : triee(L1)
Out: False
In : L2 = [2, 3, 4, 5, 5, 8, 9]
In : triee(L2)
Out: True
```

On décrit alors ci-dessous l'algorithme pouvant être mis en œuvre sur une telle liste triée.

- On commence par considérer le premier élément de cette liste triée (d'indice de position i_g) et le dernier élément de la liste (d'indice de position i_d).
- Puis, tant que i_g est strictement inférieur à i_d , on répète les opérations suivantes :
 - on calcule la somme s_{gd} des éléments de positions i_g et i_d
 - si s_{gd} est strictement inférieure à v , on incrémente i_g de 1
 - si s_{gd} est strictement supérieure à v , on décrémente i_d de 1
 - si s_{gd} est égale à v , on renvoie le tuple (i_g, i_d) .
- Si l'on sort de la boucle précédente sans avoir trouvé une paire d'entiers valide, on renvoie `False`.

Exemple d'application :

Premières étapes de
l'algorithme pour une liste
 $L = [0, 1, 3, 4, 5, 6, 9]$
avec une valeur cible $v = 8$



D.2. Ecrire une fonction `rechercheD(L_triee, v)` :

- qui prend en arguments une liste d'entiers `L_triee` (liste triée par valeurs croissantes) et un entier `v` (valeur cible)
- qui met en œuvre l'algorithme décrit ci-dessus.

/ 3 pt