

Informatique Tronc Commun
Devoir surveillé n°1
Correction et barème

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

A. Recherche aléatoire

A.1. Appel de la fonction `alea` afin de créer deux entiers `i1` et `i2` :

```
n = len(L)
i1, i2 = alea(0, n-1)
```

Barème :

- Création de la variable `n` (exploitation de la fonction `len`) **(0,5 pt)**
- Création des deux entiers `i1` et `i2` (appel de la fonction `alea` avec les deux arguments attendus) **(1 pt)**

A.2. Création de la variable booléenne `reponse` :

```
reponse = (L[i1] + L[i2] == v)
```

```
if L[i1] + L[i2] == v :
    reponse = True
else :
    reponse = False
```

Barème :

- Création cohérente de la variable `reponse` (distinction valeur / indice pour la liste `L`, test d'égalité) **(1 pt)**

A.3. Répétition des tirages aléatoires « tant que » la valeur cible n'est pas atteinte :

```
i1, i2 = alea(0, n-1)
while L[i1] + L[i2] != v :
    i1, i2 = alea(0, n-1)
print((i1, i2))
```

Barème :

- Initialisation des variables `i1` et `i2` (celle de `n` n'est pas imposée : cf. A.1) **(0,5 pt)**
- Boucle `while` cohérente (condition de poursuite, indentation, actualisation de `i1` et `i2`) **(1 pt)**
- Affichage du résultat attendu dans la console (doubles parenthèses impératives) **(0,5 pt)**

A.4. Limitation du nombre de tirages aléatoires :

```
i1, i2 = alea(0, n-1)
nb_tirages = 1
while L[i1] + L[i2] != v and nb_tirages < 1000 :
    i1, i2 = alea(0, n-1)
    nb_tirages = nb_tirages + 1
print(L[i1] + L[i2] == v)
```

```

for nb_tirages in range(1000) :
    i1, i2 = alea(0, n-1)
    if L[i1] + L[i2] == v :
        break
print(L[i1] + L[i2] == v)

```

Barème :

- Limitation effective du nombre maximal de tirages aléatoires **(1 pt)**
- Interruption des tirages aléatoires dès qu'un couple adéquat (i_1, i_2) est trouvé **(0,5 pt)**
- Affichage du résultat attendu dans la console **(0,5 pt)**

B. Recherche exhaustive

B.1. Sélection des indices i_2 vérifiant $i_2 < i_1$ (figure ci-contre).

On constate ainsi que $i_2 \in \llbracket 0; i_1 - 1 \rrbracket$ pour tout $i_1 \in \llbracket 1; n - 1 \rrbracket$ en considérant ici une liste L de $n = 6$ éléments.

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

Barème :

- Sélection effective de tous les indices i_2 valides **(1 pt)**

B.2. Enumération de tous les couples d'indices à l'aide de deux boucles imbriquées :

```

n = len(L) # Taille de la liste L
for i1 in range(1, n) :
    for i2 in range(0, i1) :

```

Barème :

- Mobilisation (même incorrecte) de l'énumérateur range **(0,5 pt)**
- Bornes correctes pour l'énumération de i_1 et i_2 **(0,5 + 0,5 pt)**

B.3. Ecriture de la fonction rechercheB(L, v) :

```

def rechercheB(L, v) :
    n = len(L)
    for i1 in range(1, n) :
        for i2 in range(0, i1) :
            if L[i1] + L[i2] == v :
                return (i1, i2)
    return False

```

Barème :

- Respect de la syntaxe de la fonction (structure « **def** ...(**args**) : / **return** ... » avec indentation) **(0,5 pt)**
- Mise en œuvre effective d'une recherche exhaustive (exploitation de B.2) **(0,5 pt)**
- Renvoi d'un couple (i_1, i_2) adéquat s'il existe et renvoi de False sinon **(1 pt)**

C. Filtrage de la liste

C.1. Ecriture de la fonction minimum(L) :

```
def minimum(L) :  
    # Initialisation  
    e_min = L[0]  
    # Enumération par valeur  
    for e in L : # ou L[1 :]  
        if e < e_min :  
            e_min = e  
    return e_min
```

```
def minimum(L) :  
    # Initialisation  
    e_min = L[0]  
    # Enumération par indice  
    for i in range(1, len(L)) :  
        if L[i] < e_min :  
            e_min = L[i]  
    return e_min
```

Barème :

- Enumération cohérente de tous les éléments de la liste L (0,5 pt)
- Algorithme cohérent de recherche de minimum (initialisation + actualisation) (1 pt)
- Respect de la syntaxe de la fonction et renvoi de la valeur minimale (0,5 pt)

C.2. Ecriture de la fonction filtrage(L, v) :

```
def filtrage(L, v) :  
    e_min = minimum(L)  
    # Construction de la liste L_new par ajouts successifs  
    L_new = []  
    for e in L :  
        if e <= v - e_min :  
            L_new.append(e)  
    return L_new
```

```
def filtrage(L, v) :  
    e_min = minimum(L)  
    # Construction de la liste L_new par compréhension  
    L_new = [e for e in L if e <= v - e_min]  
    return L_new
```

Barème :

- Création de la variable e_min (appel de la fonction minimum) (0,5 pt)
- Création cohérente de la liste L_new (par ajouts successifs ou par compréhension, avec restriction conditionnelle) (1 pt)
- Respect de la syntaxe de la fonction et renvoi de la nouvelle liste L_new (0,5 pt)

D. Algorithme appliqué à une liste triée

D.1. Ecriture de la fonction `triee(L)` :

```
def triee(L) :  
    # Comparaison des éléments consécutifs  
    n = len(L)  
    for i in range(0, n-1) :  
        if L[i] > L[i+1] :  
            return False  
    return True
```

```
def triee(L) :  
    reponse = True  
    # Comparaison des éléments consécutifs  
    n = len(L)  
    for i in range(0, n-1) :  
        if L[i] > L[i+1] :  
            reponse = False  
    return reponse
```

Barème :

- Comparaison d'éléments consécutifs sans dépassement d'indices **(1 pt)**
- Cohérence du contrôle du tri (renvoi de la valeur booléenne idoine, bascule éventuel de la valeur d'une variable booléenne `reponse`, respect de la syntaxe de la fonction) **(1 pt)**

D.2. Ecriture de la fonction `rechercheD(L_triee, v)` :

```
def rechercheD(L_triee, v) :  
    n = len(L)  
    i_g, i_d = 0, n-1  
    while i_g < i_d :  
        s_gd = L_triee[i_g] + L_triee[i_d]  
        if s_gd < v :  
            i_g = i_g + 1  
        elif s_gd > v :  
            i_d = i_d - 1  
        else :  
            return (i_g, i_d)  
    return False
```

Barème :

- Initialisation des variables `i_g` et `i_d` **(0,5 pt)**
- Boucle `while` (condition de poursuite, indentation) **(0,5 pt)**
- Calcul de `s_gd` à chaque itération (avec distinction entre `L_triee` et `L`) et traduction correcte de l'algorithme (structure `if / elif / else`, avec renvoi de la valeur adéquate) **(1 pt)**