

Le schéma accumulateur

● Méthode

Pour chaque fonction à écrire, appliquer la routine de la fiche méthode : **exemple à la main, entrée/sortie, schéma, code puis trace**. Travailler d'abord sur papier ; l'ordinateur ne sert qu'à vérifier. Les fonctions `sum`, `max`, `min` et `count` sont interdites.

● Exercice 1

Échauffement. Pour chaque fonction, écrire d'abord un exemple d'appel et le résultat attendu.

1. `somme(L: list) -> float` renvoie la somme des éléments de `L`.
2. `moyenne(L: list) -> float` renvoie la moyenne des éléments de la liste non vide `L`. On utilisera la fonction `somme`.
3. `nb_occurrences(L: list, v) -> int` renvoie le nombre de fois où `v` apparaît dans `L`.
4. `produit(L: list) -> float` renvoie le produit des éléments de `L`. Que doit renvoyer `produit([])` ?

Corrigé.

```
def somme(L: list) -> float:
    '''Renvoie la somme des éléments de L.'''
    s = 0
    for x in L:
        s = s + x
    return s

def moyenne(L: list) -> float:
    '''L est non vide. Renvoie la moyenne des éléments de L.'''
    return somme(L) / len(L)

def nb_occurrences(L: list, v) -> int:
    '''Renvoie le nombre d'éléments de L égaux à v.'''
    n = 0
    for x in L:
        if x == v:
            n = n + 1
    return n

def produit(L: list) -> float:
    '''Renvoie le produit des éléments de L (1 si L est vide).'''
    p = 1                # élément neutre de la multiplication
    for x in L:
        p = p * x
    return p
```

`produit([])` renvoie 1, élément neutre du produit, de même que `somme([])` renvoie 0.

● Exercice 2

Trouver l'erreur. Les quatre fonctions ci-dessous devraient toutes renvoyer la somme des éléments de `L`. Pour chacune, calculer à la main ce que donne l'appel sur `[4, 1, 2]`, expliquer l'erreur et corriger le code.

```
def somme1(L):
    for x in L:
        s = 0
        s = s + x
    return s
```

```
def somme2(L):
    s = 0
    for x in L:
        s = s + x
    return s
```

```
def somme3(L):
    s = 0
    for x in L:
        s = s + x
    print(s)
```

```
def somme4(L):
    s = 0
    for x in L:
        s = s + L[x]
    return s
```

Pour `somme3`, que se passe-t-il si l'on écrit ensuite `m = somme3([4, 1, 2]) / 3`?

Corrigé.

- `somme1` renvoie 2 : `s` est remis à 0 à chaque tour, il ne reste que le dernier élément. L'initialisation doit précéder la boucle.
- `somme2` renvoie 4 : le `return` est dans la boucle, la fonction s'arrête dès le premier élément. Il faut désindenter le `return`.
- `somme3` affiche 7 mais renvoie `None` ; le calcul `somme3([4, 1, 2]) / 3` provoque une `TypeError`. Remplacer `print(s)` par `return s`.
- `somme4` : `x` vaut successivement 4, 1, 2 ; `L[4]` n'existe pas, d'où une `IndexError`. Écrire `s = s + x`, ou parcourir par indice avec `for i in range(len(L))` et `L[i]`.

● Exercice 3

Altitudes d'une randonnée (d'après CCINP PC 2025).

Un itinéraire est une liste de points ; chaque point est un triplet de flottants (`latitude`, `longitude`, `altitude`), l'altitude étant en mètres. On rappelle qu'un triplet se dépaquette comme ceci : (`a`, `b`, `c`) = `p`. On considère la fonction suivante, dont l'argument est non vide :

```
def mystere(iti: list) -> float:
    s = 0
    for i in range(len(iti)):
        (lat, long, alt) = iti[i]
        s = s + alt
    return s / len(iti)
```

1. Que renvoie `mystere` appliquée à la liste formée des quatre points (47.8741, 1.8758, 100), (47.8744, 1.8759, 102), (47.8748, 1.8761, 110) et (47.8750, 1.8759, 108) ? Quelle est la signification de cette valeur ?
2. Donner la complexité de `mystere` en fonction du nombre n de points.
3. Réécrire `mystere` avec un parcours par valeur, en lui donnant un nom explicite et une spécification.
4. Écrire une fonction `nb_points_au-dessus(iti: list, h: float) -> int` qui renvoie le nombre de points de l'itinéraire dont l'altitude est strictement supérieure à `h`.
5. En déduire une fonction `proportion_au-dessus(iti: list, h: float) -> float` qui renvoie la proportion (entre 0 et 1) de ces points.

Corrigé.

1. $(100 + 102 + 110 + 108)/4 = 105$: c'est l'altitude moyenne des points de l'itinéraire, en mètres.
2. La boucle effectue n tours, chacun en temps constant : complexité $O(n)$.
3. Version par valeur :

```
def altitude_moyenne(iti: list) -> float:
    '''iti : liste non vide de triplets (lat, long, alt).
    Renvoie l'altitude moyenne des points de iti.'''
    s = 0
```

```

for (lat, long, alt) in iti:
    s = s + alt
return s / len(iti)

```

4. et 5.

```

def nb_points_au-dessus(iti: list, h: float) -> int:
    '''Renvoie le nombre de points de iti d'altitude strictement
    supérieure à h.'''
    n = 0
    for (lat, long, alt) in iti:
        if alt > h:
            n = n + 1
    return n

def proportion_au-dessus(iti: list, h: float) -> float:
    '''iti est non vide. Renvoie la proportion de points de iti
    d'altitude strictement supérieure à h.'''
    return nb_points_au-dessus(iti, h) / len(iti)

```

● Exercice 4

Distance entre deux spectres (d'après CCINP PC 2026).

Un spectre est représenté par une liste de N flottants. La distance entre deux spectres S_1 et S_2 est définie par

$$d(S_1, S_2) = \sqrt{\frac{1}{N} \sum_{i=0}^{N-1} (S_{1,i} - S_{2,i})^2}.$$

1. Calculer à la main $d(S_1, S_2)$ pour $S1 = [3, 2, 5, 1]$ et $S2 = [1, 4, 3, 3]$.
2. Pourquoi un parcours par valeur `for x in S1`: ne suffit-il pas ici ?
3. Écrire une fonction `distance(S1: list, S2: list) -> float` qui renvoie la distance entre deux spectres de même longueur. On pourra utiliser `x ** 0.5` pour calculer une racine carrée.
4. Donner sa complexité en fonction de N .

Corrigé.

1. Les écarts valent $2, -2, 2, -2$, leurs carrés 4; la somme vaut 16, divisée par $N = 4$ elle donne 4, d'où $d = 2$.
2. On doit accéder *au même rang* aux deux listes : il faut l'indice i pour lire `S1[i]` et `S2[i]`.

```

3. def distance(S1: list, S2: list) -> float:
    '''S1, S2 : listes de flottants de même longueur N > 0.
    Renvoie la distance quadratique moyenne entre S1 et S2.'''
    N = len(S1)
    s = 0 # somme des carrés des écarts
    for i in range(N):
        s = s + (S1[i] - S2[i]) ** 2
    return (s / N) ** 0.5

```

4. N tours de boucle en temps constant : $O(N)$.

● Exercice 5

Pour aller plus loin : histogramme des lignes d'une image (d'après CCINP PC 2023).

Une image binarisée est une liste de listes : `im[i][j]` est le pixel de la ligne i et de la colonne j ; il vaut 0 (noir) ou 255 (blanc). Pour repérer les lignes de texte d'un document scanné, on compte les pixels noirs de chaque ligne de l'image.

1. Pour `im = [[255, 0, 0, 255], [255, 255, 255, 255], [0, 255, 0, 0]]`, quel doit être le résultat ?

2. Écrire une fonction `nb_noirs(ligne: list) -> int` qui renvoie le nombre de pixels noirs d'une ligne.
3. En déduire une fonction `histo_lignes(im: list) -> list` qui renvoie la liste des nombres de pixels noirs de chaque ligne.
4. Réécrire `histo_lignes` sans utiliser `nb_noirs`. Où doit se trouver l'initialisation du compteur ?

Corrigé.

1. [2, 0, 3].
2. et 3.

```
def nb_noirs(ligne: list) -> int:
    '''Renvoie le nombre de pixels valant 0 dans ligne.'''
    n = 0
    for pix in ligne:
        if pix == 0:
            n = n + 1
    return n

def histo_lignes(im: list) -> list:
    '''im : image binarisée (liste de listes de 0 et 255).
    Renvoie la liste des nombres de pixels noirs de chaque ligne.'''
    histo = []
    for ligne in im:
        histo.append(nb_noirs(ligne))
    return histo
```

4. Le compteur doit être remis à zéro au début de chaque ligne, donc entre les deux boucles :

```
def histo_lignes(im: list) -> list:
    '''Même spécification que précédemment.'''
    histo = []
    for ligne in im:
        n = 0 # nouveau compteur pour chaque ligne
        for pix in ligne:
            if pix == 0:
                n = n + 1
        histo.append(n) # une fois la ligne entièrement parcourue
    return histo
```

● Méthode

Travail personnel : deux exercices flash (15 minutes chacun).

1. `nb_voyelles(mot: str) -> int` renvoie le nombre de voyelles (sans accent) de la chaîne `mot`. On pourra tester `c in 'aeiouy'`.
2. (D'après CCINP PC 2026.) `coords` est une liste de triplets (`x`, `y`, `z`). Écrire `poids(coords: list, i: int, j: int) -> float` qui renvoie $(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2$. Pour `coords = [(0, 1, 5), (-1, 2, -4), (3, 2, 5)]`, `poids(coords, 0, 2)` doit renvoyer 10.

Corrigé.

```
def nb_voyelles(mot: str) -> int:
    '''Renvoie le nombre de voyelles non accentuées de mot.'''
    n = 0
    for c in mot:
        if c in 'aeiouy':
            n = n + 1
    return n
```

```
def poids(coords: list, i: int, j: int) -> float:
    '''coords : liste de triplets (x, y, z).
    Renvoie le carré de la distance entre les points i et j.'''
    s = 0
    for k in range(3):          # les 3 coordonnées
        s = s + (coords[i][k] - coords[j][k]) ** 2
    return s
```