Python: Compter et cumuler des valeurs dans une boucle

Lorsqu'on utilise une boucle, on ne fait pas que répéter des instructions. Très souvent, on veut aussi **compter des éléments** ou **additionner des valeurs** au fil des répétitions. Cet article explique comment fonctionnent ces mécanismes, comment les coder correctement, et surtout comment **raisonner** pour éviter les erreurs.

5GMS 5TTR 6TTR 3TTR Exploration

Pourquoi compter ou cumuler dans une boucle ?

Une boucle `for` exécute un bloc de code **plusieurs fois**. Il est donc naturel de vouloir :

- **compter** combien de fois quelque chose se produit (ex. : nombre de lettres, nombre de valeurs, nombre d'essais) ;
- **cumuler** des valeurs (ex. : somme de nombres, total de points, score final).

👉 Ces deux idées reposent sur **le même principe fondamental**.

Le principe clé : une variable qui évolue

Pour compter ou additionner, on utilise toujours une **variable qui change de valeur** à chaque itération.

Cette variable doit :

1. être **initialisée avant la boucle** ;
2. être **modifiée à chaque tour de boucle**.

Compter : augmenter de 1 à chaque fois

Exemple simple : compter jusqu'à 5

```
count = 0

for i in range(5):
    count = count + 1

print(count)
```

Ce qui se passe réellement

- `count` commence à `0`
- À chaque passage dans la boucle, on ajoute `1`
- Après 5 passages, `count` vaut `5`

👉 La ligne importante est :

```
count = count + 1
```

On peut la lire comme :

> *« le nouveau `count` devient l'ancien `count` plus 1 »*

Cumuler : additionner une valeur qui change

Exemple : somme des nombres de 1 à 5

```
total = 0

for i in range(1, 6):
    total = total + i

print(total)
```

Interprétation

- `total` commence à `0`
- `i` prend successivement les valeurs `1`, `2`, `3`, `4`, `5`
- À chaque tour, on ajoute `i` au total

👉 Ici, la ligne clé est :

```
total = total + i
```

Cela signifie :

> *« j'ajoute la valeur actuelle de `i` au total existant »*

Différence entre compter et cumuler

Objectif	Variable modifiée	Ce qu'on ajoute
Compter	`count`	toujours `+1`
Cumuler	`total`	une valeur (`i`, un score, un prix, etc.)

Erreur fréquente à éviter ❌

Réinitialiser la variable dans la boucle

```
for i in range(5):
    count = 0      # ❌ erreur
    count = count + 1
```

➡️ Ici, `count` est remis à `0` à chaque tour. Résultat final : `count` vaut toujours `1`.

👉 **L'initialisation doit toujours être faite avant la boucle.**

Astuce essentielle : raisonner sur papier 🖍️

Quand un programme devient difficile à suivre, **il faut le simuler**.

Méthode recommandée

1. Trace une **ligne par variable**

2. Fais évoluer les valeurs **itération par itération**

Exemple de simulation sur papier

Code :

```
total = 0

for i in range(1, 4):
    total = total + i
```

Tableau de suivi :

Tour	i	total avant	total après
1	1	0	1
2	2	1	3
3	3	3	6

➡️ Résultat final : `6`

👉 Cette méthode permet de :

- comprendre ce que fait réellement le programme ;
- repérer immédiatement les erreurs de raisonnement ;
- expliquer son code à quelqu'un d'autre.

Cas très courant : compter sous condition

Exemple : compter le nombre de lettres `"a"` dans un mot.

```
mot = "banane"
count = 0

for lettre in mot:
    if lettre == "a":
        count = count + 1

print(count)
```

Ici :

- la boucle parcourt chaque lettre ;
- le compteur n'augmente **que si la condition est vraie**.

À retenir

- Pour **compter**, on ajoute toujours `1` à une variable compteur.
- Pour **cumuler**, on ajoute une valeur qui change à une variable total.
- La variable de comptage ou de cumul :
  - est initialisée **avant** la boucle ;
  - est modifiée **dans** la boucle.
- En cas de doute, **simuler sur papier** avec un tableau est la meilleure stratégie.
- Comprendre l'évolution des variables est plus important que mémoriser le code.

Si tu veux, je peux ensuite :

- proposer **une série d'exercices ciblés "compter / cumuler"**,
- écrire un **article jumeau** sur les erreurs fréquentes,
- ou relier cette notion à des **cas concrets de jeu ou de statistiques**.