Les **générateurs** sont un autre concept très important en Python, souvent utilisés pour **générer des séquences d'éléments de manière paresseuse**, c'est-à-dire qu'ils produisent les éléments à la demande, un par un, au lieu de les stocker tous en mémoire. Cela permet de gagner en **performance** et de travailler avec des ensembles de données **très volumineux** sans consommer trop de mémoire.

**Concept : Les Générateurs en Python 🌟**

**1. Définition**

Un générateur est une fonction spéciale en Python qui renvoie un **itérateur**, c’est-à-dire un objet capable de produire une séquence d’éléments, mais un à un, en utilisant l’instruction yield. Contrairement aux **listes** ou autres séquences, les générateurs n'existent pas entièrement en mémoire ; ils génèrent les valeurs à la volée lorsque cela est nécessaire.

**2. Utilisation de yield**

Le mot-clé yield est utilisé dans une fonction pour convertir cette fonction en générateur. Contrairement à return, yield permet à la fonction de **suspendre son exécution** et de renvoyer une valeur, tout en **gardant son état**. Lors de l’appel suivant, la fonction reprendra son exécution là où elle s’était arrêtée.

**Exemple d'un générateur simple**

Voici un exemple simple pour comprendre le concept :

python

CopyEdit

def compteur():

i = 1

while True:

yield i

i += 1

# Créer un générateur

gen = compteur()

# Utiliser le générateur

print(next(gen)) # 1

print(next(gen)) # 2

print(next(gen)) # 3

Dans cet exemple, la fonction compteur génère des nombres à partir de 1, à l’infini, chaque fois qu'on appelle next(gen).

**Avantages des Générateurs :**

* **Mémoire efficace** : Au lieu de stocker toutes les valeurs dans une liste, les générateurs génèrent chaque valeur à la demande.
* **Performance** : En produisant les éléments à la volée, tu peux traiter des séquences très grandes, voire infinies, sans saturer la mémoire.
* **Pipelines** : Les générateurs sont souvent utilisés dans des **chaînes** de traitements, par exemple dans le cas de lecture de fichiers, traitement de données ou flux continus.

**Exemple d'utilisation dans une boucle**

Un générateur peut être utilisé directement dans une boucle, ce qui simplifie beaucoup de code. Voici un exemple où nous parcourons un générateur dans une boucle for :

python

CopyEdit

def square\_numbers(limit):

for i in range(limit):

yield i \*\* 2

# Utilisation du générateur dans une boucle

for num in square\_numbers(5):

print(num)

Sortie attendue :

CopyEdit

0

1

4

9

16

Ici, au lieu de créer une liste de carrés, le générateur crée les carrés un à un à chaque itération.