Bienvenue dans le **Concept #13 : @property et Encapsulation Avancée** — c’est l’arme secrète pour écrire du code **propre**, **sécurisé** et **intelligent** 🔐🐍

**🧠 Concept #13 — @property : Le getter moderne**

**🔹 1. Problème à résoudre :**

Parfois, tu veux **protéger un attribut** :  
✅ l’afficher  
❌ mais interdire/modérer sa modification

**🔹 2. La vieille méthode (pas très élégante) :**

python

CopyEdit

class Produit:

def \_\_init\_\_(self, prix):

self.\_prix = prix

def get\_prix(self):

return self.\_prix

def set\_prix(self, valeur):

if valeur >= 0:

self.\_prix = valeur

😒 Beaucoup de code, pas très "Pythonic".

**🔹 3. La solution propre : @property**

python

CopyEdit

class Produit:

def \_\_init\_\_(self, prix):

self.\_prix = prix

@property

def prix(self):

return self.\_prix

@prix.setter

def prix(self, valeur):

if valeur < 0:

raise ValueError("Le prix doit être positif.")

self.\_prix = valeur

✅ Utilisation :

python

CopyEdit

p = Produit(10)

print(p.prix) # ← appelle le getter

p.prix = 20 # ← appelle le setter

🎯 L’attribut **se comporte comme une variable**, mais est **entièrement contrôlé** 💡