Excellente initiative ! 🌟  
Tu arrives maintenant dans un monde très puissant de Python : la **Programmation Orientée Objet (POO)**, un style de programmation qui te permet de **modéliser des objets du monde réel**, de créer des **structures propres**, **réutilisables** et **organisées** 🧱🧠

**🧱 Partie 1 : Introduction à la Programmation Orientée Objet (POO)**

**✅ Concepts clés :**

| **Terme** | **Définition** |
| --- | --- |
| **Classe** | Le **plan** ou **modèle** de ce qu’est un objet |
| **Objet** | Une **instance** d’une classe |
| **Attributs** | Des **données** stockées dans un objet |
| **Méthodes** | Des **fonctions** qui appartiennent à une classe et manipulent ses objets |
| \_\_init\_\_ | Méthode spéciale pour **initialiser** un objet (le constructeur) |

## 🧱 Plan détaillé de la formation POO en Python :

| **🧠 Concept** | **✅ Objectif principal** |
| --- | --- |
| 1. **Classe et Objet** | Créer des objets à partir d’un plan |
| 2. **Attributs d’instance** | Stocker des données spécifiques à chaque objet |
| 3. **Méthodes** | Ajouter des comportements aux objets |
| 4. **Méthode spéciale \_\_init\_\_** | Initialiser les objets automatiquement |
| 5. **Encapsulation** | Cacher des données sensibles (\_, \_\_) |
| 6. **Méthodes spéciales** | Rendre les objets plus lisibles (\_\_str\_\_, \_\_repr\_\_, etc.) |
| 7. **Classe vs Instance** | Faire la différence entre l’objet et son plan |
| 8. **Attributs de classe** | Valeurs partagées entre tous les objets |
| 9. **Héritage** | Créer des sous-classes (classe enfant) |
| 10. **Polymorphisme** | Partager une même interface avec des comportements variés |
| 11. **Composition** | Une classe peut utiliser d’autres classes comme attributs |
| 12. **Gestion des erreurs OOP** | Exceptions dans les classes, validation, etc. |

## 🗺️ ****Plan des Concepts Bonus (POO Niveau Expert)****

| **#** | **Concept** | **Objectif principal** |
| --- | --- | --- |
| 13 | @property et encapsulation avancée | Contrôler l’accès à des attributs comme un pro 👑 |
| 14 | Abstraction avec ABC | Créer des modèles/logiques de classe imposées 🧱 |
| 15 | Duck Typing | Penser comportement plutôt que typage rigide 🦆 |
| 16 | Surcharge d’opérateurs (\_\_add\_\_, etc.) | Permettre à tes objets d’être manipulés comme des nombres 📊 |