📚 Cours : Programmation Orientée Objet avec Python

🎯 Objectif du cours :

Comprendre les fondements de la POO en Python, savoir créer et manipuler des classes et objets, et appliquer les concepts d'encapsulation, d'héritage et de polymorphisme.

---

1. 🔍 Qu'est-ce que la Programmation Orientée Objet ?

- Paradigme de programmation basé sur la **modélisation d'objets réels** sous forme de **classes**.
- Chaque **objet** a des **attributs** (état) et des **méthodes** (comportements).
- Avantages : modularité, réutilisabilité, clarté du code, maintenance facilitée.

---

2. 🧱 Classes et Objets en Python

a. Définir une classe

```
class Voiture:
    def __init__(self, marque, couleur):
        self.marque = marque
        self.couleur = couleur

    def demarrer(self):
        print(f"La {self.marque} démarre.")
```

b. Créer un objet (instance)

```
ma_voiture = Voiture("Toyota", "rouge")
ma_voiture.demarrer()
```

---

3. 🔧 Constructeur et Attributs

- `__init__()` : méthode spéciale appelée à la création d'un objet.
- `self` : référence à l'objet lui-même (équivalent de "this" dans d'autres langages).

```
class Animal:
    def __init__(self, nom):
        self.nom = nom

    def parler(self):
        print(f"{self.nom} fait un bruit.")
```

---

4. 🔄 Héritage

- Une classe peut hériter des attributs et méthodes d'une autre classe.

```
class Chien(Animal):
    def parler(self):
        print(f"{self.nom} aboie.")
mon_chien = Chien("Rex")
mon_chien.parler()
```

---

5. 🔐 Encapsulation

- Masquer l'accès direct aux données internes.
- Convention : `_protegé`, `__privé`

```
class CompteBancaire:
    def __init__(self, solde):
        self.__solde = solde  # attribut privé

    def afficher_solde(self):
        print(f"Solde : {self.__solde} €")
```

---

6. 🔄 Polymorphisme

- Capacité d'une méthode à s'adapter selon l'objet.

```
animaux = [Chien("Rex"), Animal("Bête")]

for animal in animaux:
    animal.parler()  # appelle la bonne version de parler()
```

---

7. ⚙ Méthodes spéciales (dunder methods)

- `__str__()`, `__len__()`, `__eq__()` etc.

```
class Livre:
    def __init__(self, titre):
        self.titre = titre

    def __str__(self):
        return f"Livre : {self.titre}"
```

📚 Résumé :

- La POO en Python permet de structurer votre code autour d'objets concrets.
- Concepts clés : **classes, objets, héritage, encapsulation, polymorphisme**.
- Python facilite la POO avec une syntaxe simple et intuitive.