L'Héritage (POO) en Python

📚 Définition d'une Collection en Python

En Python, une **collection** est un type de données conteneur (ou structure de données) capable de stocker et d'organiser plusieurs éléments. 📦 Les collections permettent de manipuler des groupes de données de manière efficace.

Les quatre types de collections natives (built-in) les plus courantes et fondamentales sont :

Type de Collection	Délimiteur	Mutabilité	Ordre (depuis Python 3.7)	Rôle et Caractéristique
Listes	Crochets []	Mutable (modifiable)	Ordonné	Stocke des éléments de types différents. L'accès se fait par indice entier (position).
Tuples	Parenthèses ()	Immutable (non-modifiable)	Ordonné	Utilisé pour stocker des données qui ne doivent pas changer (ex: coordonnées géographiques).
Dictionnaires	Accolades {}	Mutable	Ordonné	Stocke des paires **clé-valeur**. L'accès se fait par une clé unique et immuable.
Sets(Ensembles)	Accolades {}	Mutable	Non-ordonné	Stocke des éléments **uniques**. Utilisé pour les opérations ensemblistes (union, intersection).

🧬 Exercice Pratique : L'Héritage (POO)

Nous allons créer une hiérarchie de classes pour modéliser des **Véhicules**.

Objectif

Comprendre et appliquer le principe d'**Héritage** pour la réutilisation de code et la spécialisation.

Partie 1 : La Super-Classe (Parent) `Vehicule`

Créez la classe générique `Vehicule`.

1. **Constructeur (`__init__`) :** Accepte deux arguments : `marque` et `modele`.
2. **Attributs :** Stockez `marque`, `modele` et ajoutez un attribut `moteur_allume` initialisé à `False`.
3. **Méthodes :**
    - `demarrer()` : Affiche un message indiquant que le véhicule démarre et met `moteur_allume` à `True`.
    - `arreter()` : Affiche un message indiquant que le véhicule s'arrête et met `moteur_allume` à `False`.

Partie 2 : Les Sous-Classes (Enfants)

Créez deux sous-classes qui héritent de `Vehicule`.

A. Classe `Voiture`

1. **Héritage :** La classe `Voiture` **hérite** de `Vehicule`.
2. **Constructeur (`__init__`) :**
    - Accepte `marque`, `modele` et `nombre_portes`.
    - Utilisez `super().__init__(marque, modele)` pour initialiser les attributs du parent.
    - Initialisez l'attribut propre : `self.nombre_portes`.
3. **Méthode `klaxonner()` :** Affiche un message spécifique : "Tuut tuut ! ».

B. Classe `Moto`

1. **Héritage :** La classe `Moto` **hérite** de `Vehicule`.
2. **Surcharge de Méthode (Overriding) :** Créez une méthode `demarrer()` spécifique à la moto :
    - Si le moteur est déjà allumé, affichez "La moto vrombit déjà.".

- Sinon, appelez la méthode `demarrer()` du parent en utilisant `super().demarrer()` et affichez en plus "Le moteur vrombit fort !".

Partie 3 : Test et Utilisation

1. Instanciez un objet `Voiture` et un objet `Moto`.
2. Démarrez la voiture, faites-la klaxonner, puis arrêtez-la.
3. Démarrez la moto, puis essayez de la démarrer une seconde fois (pour tester la surcharge).
4. Arrêtez la moto.