Programmation Orientée Objet (POO) en Python

La programmation orientée objet (POO) est un paradigme de programmation largement utilisé dans lequel les programmes sont structurés autour d'objets qui représentent des entités réelles ou conceptuelles. En Python, la POO est mise en œuvre à l'aide de classes et d'objets.

Classes et Objets

- **Classe** : Une classe est un modèle pour créer des objets. Elle définit les propriétés et les comportements que les objets auront. Par exemple, une classe Animal pourrait définir les propriétés comme nom et âge, ainsi que des comportements comme manger et dormir.
- **Objet** : Un objet est une instance d'une classe. Il représente une occurrence spécifique de cette classe et possède ses propres valeurs pour les attributs de la classe. Par exemple, un objet Chien pourrait être une instance de la classe Animal, avec un nom spécifique comme "Rex" et un âge spécifique comme 3 ans.

Encapsulation, Héritage et Polymorphisme

- **Encapsulation** : L'encapsulation est le principe de regrouper les données (attributs) et les méthodes (fonctions) qui les manipulent dans une seule unité, c'est-à-dire la classe. Cela permet de cacher les détails d'implémentation et de protéger les données sensibles.
- **Héritage** : L'héritage permet à une classe (appelée classe enfant ou sous-classe) de hériter des attributs et des méthodes d'une autre classe (appelée classe parente ou super-classe). Cela favorise la réutilisabilité du code et permet de créer des hiérarchies de classes.
- **Polymorphisme** : Le polymorphisme permet à des objets de différentes classes de répondre de manière similaire à des messages ou à des appels de méthode. Cela permet d'écrire du code générique qui peut fonctionner avec différentes types d'objets

Exemple :

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

    def manger(self):
        print(f"{self.nom} mange.")

class Chien(Animal):
    def aboyer(self):
        print(f"{self.nom} aboie.")

chien1 = Chien("Rex", 5)
chien1.manger()  # Output: Rex mange.
chien1.aboyer()  # Output: Rex aboie.
```

Dans cet exemple, `Animal` est la classe parente avec une méthode `manger`, et `Chien` est une classe enfant qui hérite de `Animal`. `Chien` ajoute sa propre méthode `aboyer`.

Exemple d'héritage, getter et setter

```
class Animal:
    def __init__(self, nom, age):
        self._nom = nom  # Attribut protégé
        self._age = age

    # Getter pour l'attribut nom
    def get_nom(self):
        return self._nom

    # Setter pour l'attribut nom
    def set_nom(self, nom):
        self._nom = nom

    # Méthode commune à tous les animaux
    def faire_du_bruit(self):
        pass

# Classe enfant héritant de la classe Animal
class Chien(Animal):
    def __init__(self, nom, age, race):
        super().__init__(nom, age)
        self._race = race

    # Redéfinition de la méthode faire_du_bruit
    def faire_du_bruit(self):
        return "Woof !"

    # Getter pour l'attribut race
    def get_race(self):
        return self._race

    # Setter pour l'attribut race
    def set_race(self, race):
        self._race = race

# Création d'une instance de la classe Chien
chien1 = Chien("Rex", 5, "Labrador")

# Utilisation du getter pour l'attribut nom
print(chien1.get_nom())  # Output: Rex

# Utilisation du setter pour l'attribut nom
chien1.set_nom("Buddy")
print(chien1.get_nom())  # Output: Buddy

# Utilisation du getter pour l'attribut race
```

```
print(chien1.get_race())  # Output: Labrador

# Utilisation du setter pour l'attribut race
chien1.set_race("Golden Retriever")
print(chien1.get_race())  # Output: Golden Retriever

# Appel de la méthode faire_du_bruit
print(chien1.faire_du_bruit())  # Output: Woof !
```

Dans cet exemple, la classe `Animal` définit un attribut protégé `_nom`, ainsi que des getters et des setters pour cet attribut. La classe enfant `Chien` hérite de `Animal` et définit un attribut```_race, ainsi que ses propres getters et setters. Ensuite, nous créons une instance de` Chienappeléechien1`` et utilisons les méthodes getters et setters pour accéder et modifier les attributs.