Variables de classe et méthodes de classe

/home/phil/Documents/LMG/STS_A_2025-2027/PYTHON_STS2/C_classe_(heritage)/02_heritage_introduction

Table des matières

Variables de classe et méthodes de classe....1
1. Introduction aux attributs de classe (variables de classe)....1
2. Introduction aux méthodes de classe....2
3. Exercices....2
4. Synthèse....2

1. Introduction aux attributs de classe (variables de classe)

Jusqu'à présent, nous avons défini des attributs **d'instance** : chaque objet possède ses propres valeurs (`mon_toutou.nom`, `mon_chat.couleur`). Mais il est parfois utile de définir un attribut **partagé par toutes les instances** d'une classe. On parle alors d'**attribut de classe** (ou variable de classe).

a. Exemple : compter les animaux créés

Imaginons que nous souhaitions savoir combien d'animaux ont été créés au total dans notre programme.

```
class Animal:
    # Attribut de classe : partagé par TOUTES les instances
    nombre_animaux = 0

    def __init__(self, nom):
        self.nom = nom
        # On incrémente l'attribut de classe à chaque création
        Animal.nombre_animaux += 1

    def faire_du_bruit(self):
        return f"{self.nom} fait un bruit."
```

Testons :

```
mon_toutou = Chien("Medor")
mon_chat = Chat("Titi", "Gris")

print(Animal.nombre_animaux)   # Affiche : 2
print(Chien.nombre_animaux)    # Affiche : 2  (hérité)
print(Chat.nombre_animaux)     # Affiche : 2  (hérité)
```

💡 **Points importants :**

- `nombre_animaux` est défini **dans la classe**, pas dans `__init__`.
- Il est accessible via la classe (`Animal.nombre_animaux`) **ou** via une instance (`mon_toutou.nombre_animaux`).
- Il est **partagé** : modifier `Animal.nombre_animaux` affecte toutes les instances.
- L'incrément se fait avec `Animal.nombre_animaux += 1` (et **non** `self.nombre_animaux += 1`, sinon Python crée un attribut d'instance qui masque l'attribut de classe).

b. ⚠️ Piège classique

```
mon_toutou.nombre_animaux = 99   # Crée un attribut d'INSTANCE sur mon_toutou
print(mon_toutou.nombre_animaux) # 99
print(Animal.nombre_animaux)     # 2  (inchangé)
```

2. Introduction aux méthodes de classe

Une **méthode de classe** est une méthode qui reçoit la classe elle-même en premier paramètre (par convention `cls`) au lieu de l'instance (`self`). On la décore avec `@classmethod`.

Elle est utile pour accéder ou modifier les attributs de classe

a. Exemple : une méthode de classe pour compter

```
class Animal:
    nombre_animaux = 0

    def __init__(self, nom):
        self.nom = nom
        Animal.nombre_animaux += 1

    def faire_du_bruit(self):
        return f"{self.nom} fait un bruit."

    @classmethod
    def combien_d_animaux(cls):
        return f"Il y a {cls.nombre_animaux} animal(aux) créé(s)."
```

Test :

```
mon_toutou = Chien("Medor")
mon_chat = Chat("Titi", "Gris")

print(Animal.combien_d_animaux())  # Il y a 2 animal(aux) créé(s).
print(Chien.combien_d_animaux())   # Il y a 2 animal(aux) créé(s).
```

3. Exercices

1) Ajoutez un attribut de classe `espece = "animal"` à `Animal`. Modifiez-le dans `Chien` (`"canidé"`) et dans `Chat` (`"félidé"`). Vérifiez avec :

```
print(Animal.espece)      # animal
print(Chien.espece)       # canidé
print(Chat.espece)        # félidé
```

2) Ajoutez une méthode de classe `est_animal(cls)` qui renvoie `True` si l'attribut de classe `espece` vaut `"animal"`, sinon `False`. Testez-la sur `Animal`, `Chien` et `Chat`.

4. Synthèse

Type	Déclaration	Accès	Partagé ?
Attribut d'instance	`self.nom = …` dans `__init__`	`instance.nom`	Non (propre à chaque objet)
Attribut de classe	`nom = ...` dans le corps de la classe	`Classe.nom` ou `instance.nom`	Oui (commun à toutes les instances)
Méthode d'instance	`def methode(self):`	`instance.methode()`	Agit sur l'instance
Méthode de classe	`@classmethod` `def methode(cls):`	`Classe.methode()`	Agit sur la classe

💡 **À retenir :** les attributs et méthodes de classe sont particulièrement utiles pour gérer des **compteurs**, des **constantes partagées** à vos objets.