OOP

**Chapiter 1: Classe et objet en python**

### Cours sur les Classes et Objets en Python (pour les débutants absolus)

---

#### \*\*1. Analogie pour Comprendre "Classe" vs "Objet"\*\*

Imagine que tu veux \*\*fabriquer des cookies\*\*.

- \*\*La classe\*\* : C'est comme le \*\*moule à cookies\*\* (la forme générale, les instructions pour faire un cookie).

- \*\*L'objet\*\* : C'est \*\*chaque cookie\*\* sorti du moule (un exemple concret avec ses propres caractéristiques).

\*\*Exemple :\*\*

- Le moule (classe) dit : "Tous les cookies doivent être ronds et avoir des pépites de chocolat".

- Chaque cookie (objet) peut avoir \*\*plus ou moins de pépites\*\*, être \*\*plus ou moins cuit\*\*, etc.

---

#### \*\*2. Définir une Classe en Python\*\*

Une classe est un \*\*plan détaillé\*\* pour créer des objets. On y définit :

- Les \*\*attributs\*\* (caractéristiques, comme la couleur, l'âge).

- Les \*\*méthodes\*\* (actions, comme manger(), dormir()).

\*\*Exemple avec un Chien :\*\*

```python

# Définition de la classe "Chien"

class Chien:

# Constructeur (méthode spéciale pour initialiser l'objet)

def \_\_init\_\_(self, nom, age):

self.nom = nom # Attribut "nom"

self.age = age # Attribut "age"

# Méthode pour aboyer

def aboyer(self):

print(f"{self.nom} dit : Woof! Woof!")

```

\*\*Explication ligne par ligne :\*\*

- `class Chien:` → On crée une classe nommée "Chien".

- `def \_\_init\_\_(self, nom, age):` → Méthode appelée à la création d'un objet.

- `self` : Représente \*\*l'objet lui-même\*\* (comme "je" ou "moi").

- `nom` et `age` : Paramètres pour personnaliser l'objet.

- `self.nom = nom` → Donne un nom au chien (ex: "Rex").

- `aboyer()` → Méthode que tous les chiens peuvent exécuter.

---

#### \*\*3. Créer des Objets (Instances)\*\*

Un objet est une \*\*réalisation concrète\*\* de la classe.

\*\*Exemple :\*\*

```python

# Création de deux chiens différents

chien\_1 = Chien("Rex", 3) # chien\_1 est une "instance" de Chien

chien\_2 = Chien("Luna", 2)

# Accéder aux attributs

print(chien\_1.nom) # Affiche "Rex"

print(chien\_2.age) # Affiche 2

# Appeler une méthode

chien\_1.aboyer() # Affiche "Rex dit : Woof! Woof!"

```

---

#### \*\*4. Autre Exemple : Une Voiture\*\*

```python

class Voiture:

def \_\_init\_\_(self, marque, modele, annee):

self.marque = marque

self.modele = modele

self.annee = annee

def description(self):

return f"{self.marque} {self.modele} ({self.annee})"

# Création d'une voiture

ma\_voiture = Voiture("Tesla", "Model S", 2023)

print(ma\_voiture.description()) # Affiche "Tesla Model S (2023)"

```

---

### Exercices Pratiques (Réalistes)

#### \*\*Exercice 1 : Gestion d'une Bibliothèque\*\*

\*\*Problème :\*\*

Crée une classe `Livre` avec :

- Attributs : `titre`, `auteur`, `pages`.

- Méthode `info()` qui affiche : "Le livre [titre] de [auteur] a [pages] pages."

\*\*Solution :\*\*

```python

class Livre:

def \_\_init\_\_(self, titre, auteur, pages):

self.titre = titre

self.auteur = auteur

self.pages = pages

def info(self):

print(f"Le livre {self.titre} de {self.auteur} a {self.pages} pages.")

# Test

livre1 = Livre("Harry Potter", "J.K. Rowling", 400)

livre1.info() # Affiche "Le livre Harry Potter de J.K. Rowling a 400 pages."

```

---

#### \*\*Exercice 2 : Calcul de Surface\*\*

\*\*Problème :\*\*

Crée une classe `Rectangle` avec :

- Attributs : `longueur` et `largeur`.

- Méthodes : `aire()` (longueur \* largeur) et `perimetre()` (2\*(longueur + largeur)).

\*\*Solution :\*\*

```python

class Rectangle:

def \_\_init\_\_(self, longueur, largeur):

self.longueur = longueur

self.largeur = largeur

def aire(self):

return self.longueur \* self.largeur

def perimetre(self):

return 2 \* (self.longueur + self.largeur)

# Test

mon\_rectangle = Rectangle(5, 3)

print(mon\_rectangle.aire()) # Affiche 15

print(mon\_rectangle.perimetre()) # Affiche 16

```

---

\*\*Résumé :\*\*

- Une \*\*classe\*\* est un plan, un \*\*objet\*\* est une réalisation de ce plan.

- Le `\_\_init\_\_` initialise les attributs.

- Les \*\*méthodes\*\* sont des fonctions liées à la classe.

Tu as maintenant les bases pour créer tes propres classes ! 🚀

Chapitre 2: Attributs et Méthodes (différence entre self, méthodes d’instance vs statiques).

Je vais t'expliquer ça comme si on parlait de \*\*fabriquer des robots\*\* ! 🤖

---

### \*\*1. Analogie pour Comprendre\*\*

Imagine une \*\*usine de robots\*\* :

- \*\*Attributs\*\* : C'est ce que \*\*chaque robot possède\*\* (ex: nom, couleur, nombre de bras).

- \*\*Méthodes d'instance\*\* : Actions que \*\*chaque robot peut faire\*\* (ex: marcher, parler).

- \*\*Méthodes statiques\*\* : Actions \*\*de l'usine\*\*, pas d'un robot spécifique (ex: compter le nombre total de robots).

---

### \*\*2. Les Attributs\*\*

Ce sont les \*\*caractéristiques\*\* de l'objet. Chaque instance (objet) a ses propres attributs.

\*\*Exemple avec un Robot :\*\*

```python

class Robot:

def \_\_init\_\_(self, nom, couleur):

self.nom = nom # Attribut d'instance (unique pour chaque robot)

self.couleur = couleur # Attribut d'instance

# Création de deux robots

robot1 = Robot("Bob", "rouge")

robot2 = Robot("Alice", "bleu")

print(robot1.nom) # Affiche "Bob" (attribut de robot1)

print(robot2.couleur) # Affiche "bleu" (attribut de robot2)

```

---

### \*\*3. Méthodes d'Instance (et le `self`)\*\*

- \*\*`self`\*\* : C'est comme le mot "\*\*moi\*\*" pour le robot.

- \*\*Méthode d'instance\*\* : Une fonction qui agit \*\*sur un robot spécifique\*\* (accède à ses attributs via `self`).

\*\*Exemple :\*\*

```python

class Robot:

# [...] (\_\_init\_\_ comme avant)

def saluer(self): # Méthode d'instance (self = robot qui appelle la méthode)

print(f"Je m'appelle {self.nom} et je suis {self.couleur} !")

robot1 = Robot("Bob", "rouge")

robot1.saluer() # Affiche "Je m'appelle Bob et je suis rouge !"

```

\*\*Explication :\*\*

- Quand tu appelles `robot1.saluer()`, Python remplace `self` par `robot1`.

- Donc `self.nom` devient `robot1.nom`.

---

### \*\*4. Méthodes Statiques\*\*

- \*\*Pas de `self`\*\* : Elles n'ont pas besoin d'un robot spécifique.

- Décorateur `@staticmethod`.

- Utile pour des actions \*\*génériques\*\* (pas liées à un objet).

\*\*Exemple :\*\*

```python

class Robot:

# [...] (\_\_init\_\_ et saluer comme avant)

@staticmethod

def usine\_ouverte(): # Méthode statique (pas de self !)

print("L'usine fonctionne 24h/24 !")

# Appel DIRECT via la classe (pas besoin d'un robot)

Robot.usine\_ouverte() # Affiche "L'usine fonctionne 24h/24 !"

```

---

### \*\*Exercices Réels\*\*

#### \*\*Exercice 1 : Gestion d'Étudiants\*\*

\*\*Problème :\*\*

Crée une classe `Etudiant` avec :

- Attributs : `nom`, `note`.

- Méthode d'instance `resultat()` qui affiche "Bonjour [nom], ta note est [note]".

- Méthode statique `est\_valide(note)` qui retourne `True` si la note est entre 0 et 20.

\*\*Solution :\*\*

```python

class Etudiant:

def \_\_init\_\_(self, nom, note):

self.nom = nom

self.note = note

def resultat(self):

print(f"Bonjour {self.nom}, ta note est {self.note}")

@staticmethod

def est\_valide(note):

return 0 <= note <= 20

# Test

etudiant1 = Etudiant("Léa", 15)

etudiant1.resultat() # Affiche "Bonjour Léa, ta note est 15"

print(Etudiant.est\_valide(25)) # Affiche False (méthode statique)

```

---

#### \*\*Exercice 2 : Calculatrice\*\*

\*\*Problème :\*\*

Crée une classe `Calculatrice` avec :

- Méthode statique `addition(a, b)` qui retourne `a + b`.

- Méthode statique `multiplication(a, b)` qui retourne `a \* b`.

\*\*Solution :\*\*

```python

class Calculatrice:

@staticmethod

def addition(a, b):

return a + b

@staticmethod

def multiplication(a, b):

return a \* b

# Test

print(Calculatrice.addition(3, 5)) # Affiche 8

print(Calculatrice.multiplication(4, 6)) # Affiche 24

```

---

### \*\*Résumé\*\*

- \*\*`self`\*\* : Accède aux attributs/méthodes \*\*d'un objet spécifique\*\*.

- \*\*Méthode d'instance\*\* : Besoin d'un objet (`self`), agit sur ses données.

- \*\*Méthode statique\*\* : Pas de `self`, appelée via la classe, logique indépendante.

Tu es maintenant prêt à créer des classes avec des méthodes intelligentes ! 🚀

Quand utiliser methode instance ou methode static( static methode)