La Programmation Orientée Objet (POO) en Python

🎯 Objectifs

À la fin de cette séance, vous serez capable de :

1. Comprendre les concepts fondamentaux de la POO.
2. Définir et instancier une **Classe** en Python.
3. Créer des **Attributs** et des **Méthodes** pour une classe.
4. Comprendre les principes d'**Encapsulation**, d'**Héritage** et de **Polymorphisme**.

🕐 Durée Estimée
90 minutes (ajustable).

1. Introduction à la POO

Qu'est-ce que la POO ?

La POO est un **paradigme de programmation** qui utilise des "**objets**" pour concevoir des applications et des modèles informatiques. Au lieu de se concentrer uniquement sur les fonctions et la logique, la POO se concentre sur les **données**(attributs) et les **comportements** (méthodes) de ces objets.

Analogie : Imaginez que vous construisez une ville.

- La **programmation procédurale** (traditionnelle) vous dirait : "Construisez une fondation, puis des murs, puis un toit." (Focus sur les étapes/fonctions).
- La **POO** vous dirait : "Créez une **Maison**, créez une **Voiture**, créez une **Personne**." (Focus sur les entités/objets).

Les 4 Piliers de la POO

Pilier	Définition
Encapsulation	Cacher les détails internes et ne montrer que ce qui est nécessaire.
Abstraction	Simplifier la réalité en se concentrant sur les caractéristiques essentielles.
Héritage	Permettre à une classe (enfant) d'acquérir les propriétés d'une autre classe (parent).
Polymorphisme	Permettre à une méthode d'avoir différentes implémentations en fonction de l'objet qui l'appelle.

2. Les Classes et les Objets

🧱 La Classe : Le Plan

Une **Classe** est la **définition** ou le **modèle** (le plan) d'un objet. Elle décrit ce que l'objet sera (ses caractéristiques et ses actions).

- **Syntaxe de base en Python :**

Python

```
class NomDeMaClasse:
    # Code de la classe
    pass
```

📦 L'Objet : L'Instance

Un **Objet** (ou **Instance**) est une réalisation concrète et unique de la classe. C'est l'élément réel créé à partir du plan.

- **Créer un objet :**

Python

```
# 'chien1' et 'chien2' sont des objets (instances) de la
classe 'Chien'
chien1 = Chien()
chien2 = Chien()
```

🖌️ Attributs et Méthodes

Élément	Description	Exemple pour la classe Chien
Attribut	Une **variable** qui contient l'état ou les données de l'objet (ses caractéristiques).	nom, race, couleur
Méthode	Une **fonction** définie à l'intérieur d'une classe qui représente le comportement ou l'action de l'objet.	aboyer(), courir(), manger()

Exemple de Code (Classe et Attributs/Méthodes)

Python

```
class Chien:
    # 1. Le constructeur (Méthode spéciale pour créer
l'objet)
    def __init__(self, nom, race):
        self.nom = nom        # Attribut d'instance
        self.race = race      # Attribut d'instance

    # 2. Une méthode d'instance
    def aboyer(self):
        return f"{self.nom} fait Wouf! Wouf!"

# Instanciation de l'objet
mon_chien = Chien("Rex", "Berger Allemand")

# Accéder aux attributs
print(f"Nom : {mon_chien.nom}, Race : {mon_chien.race}")

# Appeler une méthode
print(mon_chien.aboyer())
```

Note sur `self` : La variable `self` représente l'instance de l'objet lui-même. C'est le premier argument de toute méthode d'instance en Python.

3. Les Piliers en Python (40 min)

🔒 Encapsulation et Abstraction

L'encapsulation est gérée en Python principalement par **convention**.

- **Attributs Publics (Accessibles partout) :** Nommé normalement (`self.nom`).
- **Attributs Protégés (Convention) :** Précédé d'un seul underscore (`self._race`). Indique aux développeurs de ne pas y toucher.
- **Attributs Privés (Nom Mangling) :** Précédé de deux underscores (`self.__age`). Python rend l'accès direct difficile, mais pas impossible.

Python

```
class CompteBancaire:
    def __init__(self, solde_initial):
        # Attribut "privé" par convention pour
l'encapsulation
        self.__solde = solde_initial

    # Méthode "publique" pour accéder aux données
(Abstraction)
    def deposer(self, montant):
        if montant > 0:
            self.__solde += montant

    def get_solde(self):
        return self.__solde
```

🧬 Héritage

L'**Héritage** permet de définir une nouvelle classe (classe enfant ou **sous-classe**) qui réutilise le code et les fonctionnalités d'une classe existante (classe parent ou **super-classe**).

- **Syntaxe :** `class Enfant(Parent):`
- **Utilisation de `super()` :** Appelle le constructeur ou une méthode du parent.

Python

```
# Super-classe (Classe Parent)
class Animal:
    def __init__(self, nom):
        self.nom = nom

    def respirer(self):
        return f"{self.nom} respire."

# Sous-classe (Classe Enfant) qui hérite d'Animal
class Chat(Animal):
    def __init__(self, nom, couleur):
        # Appel du constructeur de la classe Parent
        super().__init__(nom)
        self.couleur = couleur

    # Nouvelle méthode propre à la classe Chat
    def miauler(self):
        return f"{self.nom} ({self.couleur}) miaule."

mon_chat = Chat("Mistigri", "Noir")
print(mon_chat.respirer()) # Méthode héritée
print(mon_chat.miauler())  # Méthode propre
```

🔄 Polymorphisme

Le **Polymorphisme** ("plusieurs formes") signifie qu'une même **méthode** peut se comporter différemment dans des classes différentes. En Python, cela est souvent réalisé par :

1. **Surcharge de méthode (Method Overriding) :** Une sous-classe fournit sa propre implémentation d'une méthode déjà définie dans sa super-classe.

Python

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

    # Surcharge de la méthode 'respirer' du Parent (Animal)
    def respirer(self):
        return f"{self.nom} halète et respire bruyamment." #
Implémentation différente

# Exemple d'utilisation du Polymorphisme
def faire_respirer(animal):
    # Appelle la méthode 'respirer' spécifique à l'objet
passé
    print(animal.respirer())

rex = Chien("Rex")
mistigri = Chat("Mistigri", "Blanc")

faire_respirer(rex)       # Affiche l'implémentation de Chien
faire_respirer(mistigri)  # Affiche l'implémentation de Chat
(héritée d'Animal)
```

4. Exercice Pratique

Créez une classe nommée `Rectangle` avec les éléments suivants :

- **Attributs :** `longueur` et `largeur`.

- **Méthodes :**
  - `__init__` pour initialiser les attributs.
  - `calculer_surface()` qui retourne la surface (`longueur * largeur`).
  - `calculer_perimetre()` qui retourne le périmètre.

5. Conclusion

La POO en Python permet de créer des programmes **modulaires**, **flexibles** et **maintenables** en regroupant les données et les comportements au sein d'entités logiques appelées objets.

- **Classe** = Plan
- **Objet** = Réalisation du plan
- **Méthodes/Attributs** = Comportements/Données