chap 1: Introduction à la Programmation Orientée Objets (POO)

1. La programmation orientée objets

- La **POO** est un **paradigme de programmation** qui organise le code autour de **classes** et **objets** au lieu de simples fonctions et instructions.
- Une **classe** est comme un **plan** ou un **modèle** (par exemple : la classe *Voiture*).
- Un **objet** est une **instance** concrète de la classe (par exemple : *ma_voiture*).

Exemple en Python :

```
class Voiture:
    """
    Classe représentant une voiture avec une marque et un modèle.
    """
    def __init__(self, marque, modele):
        """
        Constructeur de la classe Voiture.
        Paramètres :
        - marque (str) : la marque de la voiture (ex : "Toyota")
        - modele (str) : le modèle de la voiture (ex : "Corolla")
        """
        # Initialisation des attributs de la voiture
        self.marque = marque
        self.modele = modele
    def afficher(self):
        """
        Affiche les informations de la voiture (marque et modèle).
        """
        print(f"Voiture : {self.marque} {self.modele}")
```

```
# Création d'une instance (un objet) de la classe Voiture
ma_voiture = Voiture("Toyota", "Corolla")
# Appel de la méthode afficher() pour afficher les informations de l'objet
ma_voiture.afficher()
```

2.Les avantages de la POO

Encapsulation

- Cacher les **détails internes** d'un objet et fournir une interface simple pour l'utiliser.
- Exemple : une voiture a un moteur, mais pour la conduire, on utilise seulement le volant et les pédales.

```
class CompteBancaire:
    def __init__(self, solde=0):
        self.__solde = solde   # attribut privé
    def deposer(self, montant):
        self.__solde += montant
    def consulter_solde(self):
        return self.__solde
```

Abstraction

- Ne montrer que les **fonctionnalités essentielles** sans exposer la complexité.
- Exemple : une voiture a une fonction *démarrer()*, mais l'utilisateur n'a pas besoin de savoir comment le moteur fonctionne.

- **3 Differnec entre programmation Classique et POO**

1. Programmation classique (procédurale)

- **Principe** : on écrit des **séquences d'instructions** qui s'exécutent les unes après les autres.
- Le programme est divisé en **fonctions** (ou procédures) qui effectuent des tâches précises.
- Les **données** (variables) et les **fonctions** (logique) sont séparées.
- Exemple concret :
  -
    - Tu as une variable `rayon`,
    - et une fonction `calculer_aire(rayon)` qui utilise cette donnée pour donner un résultat.

👉 C'est une logique de **"faire étape par étape"(séquentielle)**.

---

2. Programmation orientée objet (POO)

- **Principe** : on organise le programme autour de **classes** et **objets**.
- Une **classe** est un modèle (comme un plan d'architecte).
- Un **objet** est une instance de cette classe (comme une maison construite à partir du plan).
- Les **données** (appelées attributs) et les **fonctions** (appelées méthodes) sont **regroupées ensemble** dans une même classe.
- Exemple concret :

```
class Cercle:
    def __init__(self, rayon):
```

```
        self.rayon = rayon
    def aire(self):
        return 3.14 * self.rayon ** 2
c = Cercle(5)
print(c.aire())  # Résultat : 78.5
```

- Ici, le **cercle** a à la fois :
  - une **donnée** : son rayon,
  - un **comportement** : la méthode pour calculer l'aire.

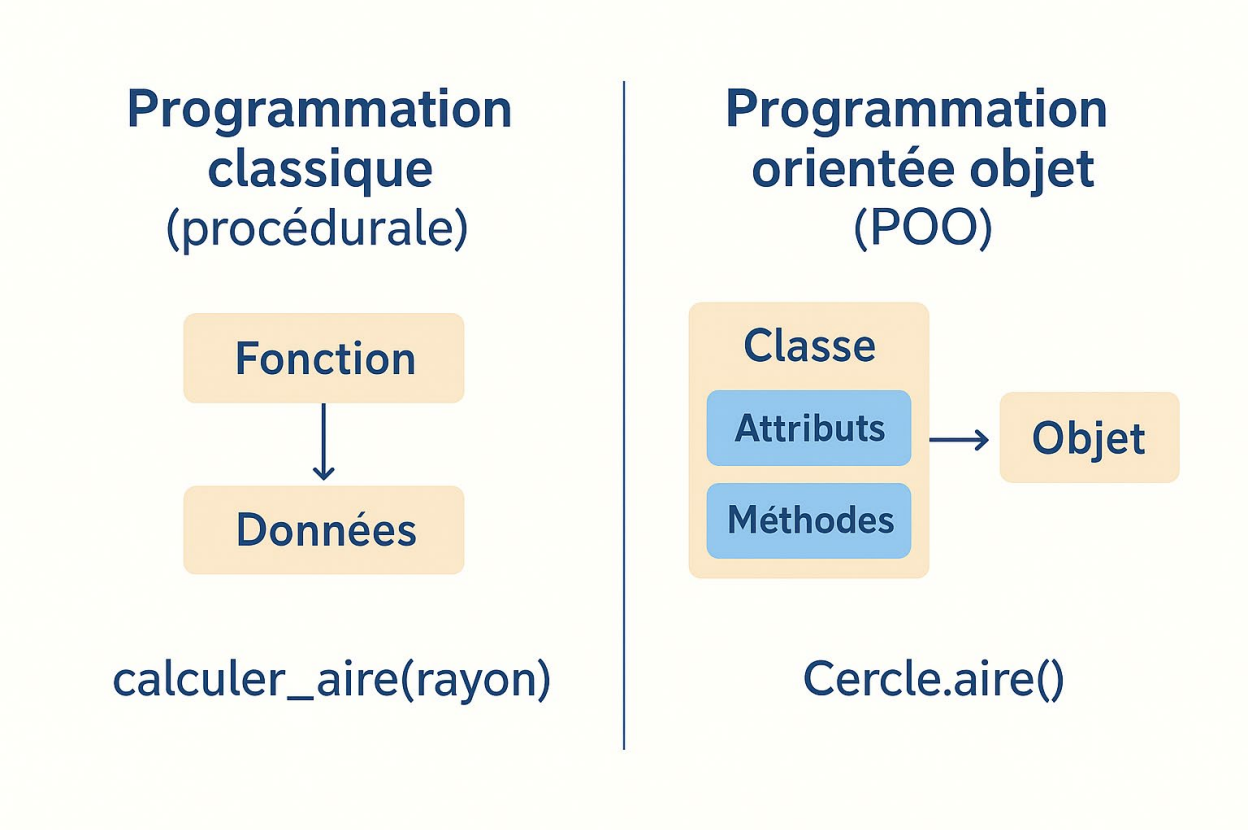
👉 C'est une logique de **"regrouper données + actions"** dans un même concept.

en résumé

Aspect	Programmation classique	Programmation orientée objet
Organisation	Fonctions + données séparées	Classes regroupant données et méthodes
Approche	Procédurale (étape par étape)	Orientée objets (modéliser le réel)
Réutilisation	Moins flexible	Plus réutilisable grâce à l'héritage et au polymorphisme
Exemple	Fonctions `calculer_aire(rayon)`	Objet `Cercle.aire()`

En clair :

- **Procédurale** = tu écris des recettes de cuisine.
- **POO** = tu définis des "chefs" (objets) qui savent cuisiner eux-mêmes leurs plats.



Initiation au langage Python

- Python est un langage simple, lisible et très utilisé.
- Syntaxe basique :

```
# Variables
nom = "Nourame"
age = 25
# Condition
if age > 18:
   print(f"{nom} est majeur")
# Boucle
```

```
for i in range(3):
    print("Bonjour")
```

Les librairies Python

- Une **librairie** est un ensemble de fonctions prêtes à l'emploi.
- Exemples :
  - `math` : pour les calculs mathématiques
  - `datetime` : pour gérer les dates et heures
  - `numpy`, `pandas` : pour les données scientifiques
  - `matplotlib` : pour tracer des graphiques

👉 Exemple :

```
import math

print(math.sqrt(16))  # permet de clculer la racine carrée
```