Une Introduction à la Programmation Orientée Objet : Les Classes et les Objets

I. Concept : Programmation Orientée Objet

Il consiste en la définition et l'interaction de briques logicielles appelées objets ; un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre. Il possède une structure interne et un comportement, et il sait interagir avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ; l'interaction entre les objets via leurs relations permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes. Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme virtuelle.

Beaucoup des types que nous découvrirons dans le prochain chapitre sont considérés comme des **Classes** en Python. Par exemple, les listes, dictionnaires, chaîne de caractère.

À la place de manipuler des Classes/types directement donnés par Python. On peut créer directement les objets/types qui nous intéressent.

Un bel exemple d'utilisation de Programmation Orientée Objet est la simulation de foule. Une vidéo de la chaîne Fouloscopie (https://www.youtube.com/watch?v=w-Oy4TYDnoQ) vulgarise bien ce concept.

II. Création d'une Classe

Pour créer une nouvelle Classe, on fait : (la nouvelle classe est Eleve)

```
1 class Eleve:
```

III. Attributs

Comme dit précédemment, **les Classes/Objets** sont utiles pour réaliser des modélisations. On va prendre le problème de modélisation suivant : on veut modéliser des

élèves de Seconde qui veulent partir en Première générale, afin de pouvoir faire un programme qui propose des répartitions d'élèves dans des classes.

Première question : Que faut-il pour caractériser un tel élève ... On posera qu'il faut connaître :

— Son nom
— Son sexe
— Ses voeux de spécialité.

Ces caractéristiques sont appelés **Attributs** de la Classe. Pour ajouter des attributs, on aura besoin d'un constructeur. Ce constructeur est la fonction/méthode : **__init__**.

```
class Eleve:
        def __init__(self):
                self.nom = "Mettre un Nom"
                self.sexe = "Adolescent"
                self.specialite = []
```

Définition 0.1.
Un objet est une instance d'une classe *i.e.* tout comme "babar" est une instance d'une chaîne de caractère. Un objet sera une instance d'une classe.

On veut créer un objet élève avec les attributs : "Kévin", "Homme", ["Math","NSI","SES"] on va affecter à une variable NouvelEleve un objet de type Eleve. et affecter à chaque attribut la valeur correspondante. Tout d'abord on crée l'objet NouvelEleve.

```
PremierEleve ← Eleve()
```

```
PremierEleve = Eleve()
```

Regardons maintenant ses attributs. Les attributs d'un objet sont des variables, don cpour les observer On entre dans la console : (les >>> représente ce que la console renvoie)

```
PremierEleve.nom
>>> "Mettre un Nom"
PremierEleve.sexe
>>> "Adolescent"
PremierEleve.specialite
>>> []
```

Pour modifier cet attribut, on entre dans la console :

```
PremierEleve ← Eleve()
PremierEleve.nom ← "Kevin"
PremierEleve.sexe ← "Homme"
PremierEleve.specialite ← ["Math","NSI","SES"]
```

On sait que **__init__** est une fonction. Donc elle prend une entrée. Ici l'entrée est *self*. Ce dernier représente l'objet.

Pour voir les attribut de l'Objet PremierEleve maintenant, j'entre de nouveau dans la console :

```
PremierEleve.nom
>>> "Kevin"
PremierEleve.sexe
>>> "Homme"
PremierEleve.specialite
>>> ["Math","NSI","SES"]
```

▪ **Exercice 0.1.**
L'objectif de cet exercice est de faire une classe Prof et de créer deux Objets : Gorce et Gibaud avec les bons attributs.

1. Trouvez les caractéristiques d'un professeur de lycée (sur papier).
2. Définir la classe Professeur (avec son constructeur)
3. Créer deux variables de type Professeur. Une variable sera Gibaud, l'autre Gorce.
4. Changer les attributs de ces deux fonctions pour que Gibaud et Gorce aient les bons attributs

▪

IV. Les Méthodes

Toute la beauté de la programmation orientée objet est que les objets ont :
— des caractéristiques appelés **Attributs**
— des actions/fonctions appelés **Méthodes**

Les méthodes sont des fonctions internes à une Classe. Cela permet aux objets d'agir et d'interagir entre eux.

Pour faire une méthode (ici DirePresent ou __init__)on entre dans la console :

```
class Eleve:
        def __init__(self):
                self.nom = "Entrer un nom"
                self.sexe = "Adolescent"
                self.voeux = []

        def DirePresent(self):
                print(self.nom + " Present !")
```

R **Toutes les méthodes prennent au moins self en entrée**

Pour appeler cette méthode, on doit déjà créer l'objet puis appeler la méthode. (on va changer l'attribut d'abord). On entre alors dans la console, l'appel de la méthode est en ligne 5 :

```
1 SecondEleve = Eleve()
2 SecondeEleve.nom = "Isma"
3 SecondEleve.sexe = "Femme"
4 SecondEleve.voeux = ["Math","NSI","HLP"]
5 SecondEleve.DirePresent()
6 >>> "Isma Present !"
```

R Pour qu'un objet appelle une méthode on met un • entre l'objet et la méthode. Comme la méthode est une fonction on met des parenthèses avec les arguments après la méthode. Si la méthode ne prend que *self* on met des parenthèses vides.

Cependant les méthodes peuvent être plus compliquées et faire des actions plus complexes. Par exemple **__init__** est une méthode ou **append** qui est une méthode pour les liste et qui permet d'ajouter un élément à la fin d'une liste.

On pourrait avoir le suivant :

```
1 class Eleve:
2         def __init__(self):
3                 self.nom = "Entrer un nom"
4                 self.sexe = "Adolescent"
5                 self.voeux = []
6         def DirePresent(self,Phrase):
7                 print(Phrase)
```

On aurait alors comme appel de DirePresent :

```
1 SecondEleve = Eleve()
2 SecondeEleve.nom = "Isma"
3 SecondEleve.sexe = "Femme"
4 SecondEleve.voeux = ["Math","NSI","HLP"]
5 SecondEleve.DirePresent("Je suis ici, Monsieur."
    )
6 >>> "Je suis ici, Monsieur"
```

On remarque cette fois DirePresent prend un argument en entrée. Cet argument est le *Phrase* de *def DirePresent(self,Phrase)*