Chapitre : Programmation Orientée Objet : Class c'est la classe

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:

In [6]:
```
class Eleve:
    pass #pass sera remplacé par des méthodes et des attributs dans la suite (qu
```

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 Cours de mathématiques Classe de Première 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__**.

In [7]:
```
class Eleve:
    def __init__(self):
        self.nom = "Mettre un nom"
        self.sexe = "Adolescent(e)"
        self.specialite = []
```

Définition : Objet
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 `premier_eleve`.

In [11]:
```
premier_eleve = Eleve()
```

Regardons maintenant ses attributs. Les attributs d'un objet sont des variables que l'on peut afficher avec `print` entre autres

In [12]:
```
print(premier_eleve.nom)
print(premier_eleve.sexe)
print(premier_eleve.specialite)
```

```
Mettre un nom
Adolescent(e)
[]
```

Pour modifier des attributs on fait :

In [ ]:
```
premier_eleve.nom = "Kévin"
premier_eleve.sexe = "Homme"
premier_eleve.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

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

- Trouvez les caractéristiques d'un professeur de lycée (sur papier).
- Définir la classe Professeur (avec son constructeur)
- Créer deux variables de type Professeur.Une variable seraGibaud, l'autre Gorce.
- 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 `dire_present` ou `__init__`) on entre dans la console :

In [1]:
```
class Eleve:
    def __init__(self):
        self.nom = "Mettre un nom"
        self.sexe = "Adolescent(e)"
        self.specialite = []
    def dire_present(self):
        print(f"{self.nom} Présent !")
    def __str__(self):
        return f"Eleve {self.nom}"
```

Remarque : 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 :

In [2]:
```
second_eleve = Eleve()
second_eleve.nom = "Isma"
second_eleve.sexe = "Femme"
second_eleve.voeux = ["Math","NSI","HLP"]
second_eleve.dire_present()
```

```
Isma Présent !
```

Afficher plus facilement les objets !!

La méthode `__str__` permet de "printer" un objet. La méthode `__str__` renvoie un string qui sera affiché quand on printera l'objet. Un petit exemple :

In [3]:
```
print(second_eleve)
```

```
Eleve Isma
```

Remarque :
Pour qu'un objet appelle une méthode on met un **.** (point) entre l'objet et la méthode. Comme la méthode est une fonction on met des parenthèses avec les arguments après la méthode. Si laméthode ne prend que self onmet 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 :

In [24]:
```
class Eleve:
    def __init__(self):
        self.nom = "Mettre un nom"
        self.sexe = "Adolescent(e)"
        self.specialite = []
    def dire_present(self,phrase):
        print(phrase)
```

On aura alors comme appel de dire présent

In [25]:
```
second_eleve = Eleve()
second_eleve.nom = "Rayan"
second_eleve.sexe = "Homme"
second_eleve.voeux = ["Math","NSI","HGGSP"]
second_eleve.dire_present("Je suis ici Monsieur, vous m'avez vu")
```

```
Je suis ici Monsieur, vous m'avez vu
```

Remarque : On remarque que cette fois `dire_present` prend un argument en entrée. Cet argument est `phrase`.

Exercice : Ajouter à la classe `Professeur` une méthode prenant en argument un texte (qui sera un texte d'exercice) et qui affiche cet exercice.

In [ ]:
```
```

Exercice :

- Faire une classe Joueur avec comme attributs un identifiant (qui sera un entier positif ou nul) et un score (qui sera aussi un entier).
- Ajouter à cette classe Joueur une méthode (vous choisirez les arguments de la méthodes) qui affiche l'identifiant et le score (dans un message lisible)
- Ajouter une méthode vol_de_point qui prend en entrée un autre Joueur et qui lui vole un nombre de points aléatoire pris entre 0 et 10. (Attention les points ne peuvent pas être négatifs)
- Faire une classe Groupe_de_Joueurs dont l' attribut est une liste de Joueurs et qui aura comme méthode : trier_la_liste (qui tri les joueurs par nombre de points croissant) et une méthode voler_dans_tous_les_sens (qui prendra en argument n et qui fera tourner n vol entre joueurs et qui triera ensuite la liste des joueurs)

In [ ]:
```
```

In [ ]:
```
```