Chapitre 0
POO

Ce chapitre est le premier chapitre de l'année de Terminale NSI. Il introduit la **programmation orientée objet** (POO) : une manière d'organiser un programme autour d'**objets**, qui regroupent des données et des actions. Nous verrons comment définir une **classe**, créer des objets, leur donner des **attributs** (leurs caractéristiques) et des **méthodes** (leurs actions).

I. La programmation orientée objet

1. Le concept d'objet

La programmation orientée objet 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 : une voiture, une personne, 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 de réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes.

Exemple 0.1.
Une voiture peut être vue comme un objet : elle possède des caractéristiques (sa marque, sa couleur, sa vitesse) et des actions (démarrer, accélérer, freiner). En programmation orientée objet, ces caractéristiques et ces actions sont regroupées dans un même objet : les caractéristiques sont ses **attributs**, les actions ses **méthodes**.

R 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. Avant d'écrire la moindre ligne de code, on réfléchit aux objets du problème, à leurs caractéristiques et à leurs actions.

2. Des types Python personnalisés

Beaucoup des types que nous utilisons depuis la Première sont en réalité des **classes** en Python : par exemple les listes, les dictionnaires, les chaînes de caractères.

Exemple 0.2.

Les méthodes que l'on utilise sur les types de Python sont des méthodes de classes : `append` est une méthode de la classe `list`, `upper` est une méthode de la classe `str`. ■

```
[1, 2, 3].append(4)
"bonjour".upper()
```

À la place de manipuler des classes et des types directement donnés par Python, on peut créer directement les objets et les types qui nous intéressent : c'est l'objet des sections suivantes.

■ **Exemple 0.3.**
Un bel exemple d'utilisation de la programmation orientée objet est la **simulation de foule** : chaque piéton est un objet qui possède des caractéristiques (position, vitesse) et des comportements (se déplacer, éviter un obstacle). 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 utilise le mot-clé `class` suivi du nom de la classe. Par convention, le nom d'une classe commence par une majuscule (on dit qu'il suit la convention *CamelCase*).

Définition 0.1.
Une **classe** est un modèle (un moule) qui décrit la structure et le comportement communs à un ensemble d'objets. Elle regroupe des **attributs** (les caractéristiques des objets) et des **méthodes** (leurs actions). Un objet est fabriqué à partir de la classe : on dit qu'il est une **instance** de cette classe.

■ **Exemple 0.4.**
La classe `Eleve` est créée avec le mot-clé `class`. Elle ne contient pour l'instant rien : le mot-clé `pass` est une instruction qui ne fait rien, utilisée pour laisser la classe vide sans provoquer d'erreur. ■

```
class Eleve:
    pass # pass sera remplace par des methodes et des attributs dans la
        suite
```

R Une classe ne représente pas un objet particulier : `Eleve` ne décrit pas un élève en particulier, mais ce que sera *tout* objet de type `Eleve`. C'est le rôle de la création d'objets (que l'on verra plus bas) d'obtenir des élèves concrets à partir de ce modèle.

III. Les attributs

1. Modéliser un problème : l'exemple des élèves

Comme dit précédemment, les classes et les objets sont utiles pour réaliser des modélisations. On prend 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 vœux de spécialité.

R Ces caractéristiques sont appelées **attributs de la classe** : ce sont les variables que chaque objet de la classe possédera en propre.

2. Le constructeur `__init__`

Pour ajouter des attributs à une classe, on a besoin d'un **constructeur**. Ce constructeur est la méthode `__init__` : elle est appelée automatiquement à la création de chaque objet, et elle initialise les attributs de l'objet.

▪ Exemple 0.5.
La classe `Eleve` et son constructeur : les attributs sont créés avec `self.` suivi du nom de l'attribut, et reçoivent des valeurs par défaut. ▪

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

À la création d'un objet de type `Eleve`, les trois attributs `nom`, `sexe` et `specialite` sont initialisés avec les valeurs par défaut ci-dessus.

Définition 0.2.
Un **objet** est une instance d'une classe. Tout comme `"babar"` est une instance de la classe `str` (une chaîne de caractères), un objet est une instance de sa classe. Créer un objet à partir d'une classe s'appelle **instancier** la classe.

3. Créer un objet et accéder à ses attributs

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

■ **Exemple 0.6.**
Créer un objet, puis afficher ses attributs. La variable `premier_eleve` contient désormais un objet de type `Eleve`, distinct de la classe : c'est une instance de la classe `Eleve`. ■

```
premier_eleve = Eleve()
```

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

```
print(premier_eleve.nom)
print(premier_eleve.sexe)
print(premier_eleve.specialite)
```

Ce programme affiche les valeurs par défaut définies dans le constructeur : `Mettre un nom`, `Adolescent(e)` et `[]`.

4. Modifier un attribut

Pour modifier un attribut, on lui affecte une nouvelle valeur avec le signe `=`, comme pour une variable classique.

■ **Exemple 0.7.**
On donne à l'objet `premier_eleve` les attributs de l'élève Kévin. ■

```
premier_eleve.nom = "Kevin"
premier_eleve.sexe = "Homme"
premier_eleve.specialite = ["Math", "NSI", "SES"]
```

L'objet `premier_eleve` a maintenant pour attributs : `"Kevin"`, `"Homme"` et `["Math", "NSI", "SES"]`.

> **R** On sait que `__init__` est une fonction : elle prend donc des entrées. Ici, l'entrée est `self`. Ce dernier représente l'objet en cours de création ou de manipulation : c'est grâce à `self` que la méthode sait sur quel objet elle travaille. C'est pour cela que, dans la classe, on écrit `self.nom` et pas simplement `nom`.

■ **Exemple 0.8.**
Deux objets créés à partir de la même classe sont indépendants : modifier les attributs de l'un ne modifie pas l'autre. ■

```
eleve1 = Eleve()
eleve2 = Eleve()
eleve1.nom = "Aya"
eleve2.nom = "Noe"
print(eleve1.nom)
print(eleve2.nom)
```

Ce programme affiche `Aya` puis `Noe` : chaque objet possède ses propres attributs.

IV. Les méthodes

1. Des fonctions dans la classe

Toute la beauté de la programmation orientée objet est que les objets ont :
— des caractéristiques appelées **attributs** ;
— des actions et des fonctions appelées **méthodes**.

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

Définition 0.3.
Une **méthode** est une fonction définie à l'intérieur d'une classe. Elle décrit une action réalisable par les objets de cette classe. Comme toutes les méthodes s'appliquent à un objet, elles prennent au moins `self` en paramètre, qui désigne l'objet sur lequel la méthode est appelée.

■ **Exemple 0.9.**
Une autre classe que `Eleve` : la classe `Chien`. Chaque chien a un nom et un âge, et il sait aboyer. ■

```
class Chien:
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age

    def aboyer(self):
        print("Wouf !")
```

On peut alors créer un chien et lui faire faire des actions :

```
rex = Chien("Rex", 3)
rex.aboyer()
print(rex.nom)
```

■ **Exemple 0.10.**
La classe `Eleve` complétée avec deux méthodes : `dire_present`, qui affiche un message de présence, et `__str__`, dont le rôle est expliqué plus bas. ■

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

    def dire_present(self):
        print(f"{self.nom} Present !")

    def __str__(self):
        return f"Eleve {self.nom}"
```

R Toutes les méthodes prennent au moins `self` en entrée. Dans la méthode `dire_present`, `self.nom` désigne l'attribut `nom` de l'objet sur lequel la méthode est appelée.

2. Appeler une méthode

Pour appeler une méthode, on doit d'abord créer l'objet, puis appeler la méthode sur cet objet. On change d'abord les attributs de l'objet, puis on appelle la méthode (l'appel est la dernière ligne du programme) :

Exemple 0.11.
Créer un objet, modifier ses attributs, puis appeler la méthode `dire_present`. ■

```
second_eleve = Eleve()
second_eleve.nom = "Isma"
second_eleve.sexe = "Femme"
second_eleve.specialite = ["Math", "NSI", "HLP"]
second_eleve.dire_present()
```

Ce programme affiche : `Isma Present !`

R 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 le nom de la méthode. Si la méthode ne prend que `self`, on met des parenthèses vides.

3. Afficher un objet : la méthode `__str__`

Afficher plus facilement les objets ! La méthode `__str__` permet d'« afficher » un objet : elle renvoie une chaîne de caractères qui sera affichée quand on affichera l'objet avec `print`.

Exemple 0.12.
Afficher l'objet `second_eleve` : c'est la chaîne renvoyée par `__str__` qui est affichée.
■

```
print(second_eleve)
```

Ce programme affiche : `Eleve Isma`. Sans la méthode `__str__`, Python afficherait une représentation par défaut peu lisible, par exemple `__main__.Eleve object at 0x7f8a2b3c4d5e`.

4. Des méthodes avec des arguments

Les méthodes peuvent être plus compliquées et faire des actions plus complexes. `__init__` est une méthode ; `append`, qui permet d'ajouter un élément à la fin d'une liste, est une méthode de la classe `list`. Une méthode peut recevoir d'autres arguments que `self`, comme une fonction classique.

▪ Exemple 0.13.
La méthode `dire_present` reçoit ici un argument : `phrase`. ▪

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

L'appel de la méthode se fait alors avec l'argument entre parenthèses :

▪ Exemple 0.14.
Appel de `dire_present` avec un argument. ▪

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

Ce programme affiche : `Je suis ici Monsieur, vous m'avez vu`.

> **R** Cette fois, `dire_present` prend un argument en entrée : cet argument est `phrase`. Lors de l'appel, la valeur passée entre parenthèses est reçue dans le paramètre `phrase`.

Pour résumer

Le tableau suivant récapitule le vocabulaire de la programmation orientée objet.

Terme	Rôle	Exemple
`Classe`	Modèle qui décrit un type d'objets	`Eleve`
`Objet`	Instance d'une classe	`premier_eleve`
`Attribut`	Caractéristique d'un objet	`self.nom`
`Méthode`	Action réalisable par un objet	`dire_present()`

V. Exercices

Les exercices suivants reprennent et complètent les notions du chapitre. Pensez à bien lire les énoncés, à écrire des programmes propres et à tester vos classes en créant des objets.

Exercices d'automatismes

Les exercices d'automatismes exercent les réflexes fondamentaux : vocabulaire de la POO, lecture de code court, prédiction d'affichage, complétion de code. Chaque question doit pouvoir être traitée en moins de trente secondes.

Exercice 0.1.

Dire si les affirmations suivantes sont vraies ou fausses.

1. Un objet est une instance d'une classe.
2. Une classe est un objet comme les autres.
3. Une méthode est une fonction interne à une classe.
4. Toutes les méthodes prennent au moins `self` en entrée.
5. Deux objets créés à partir de la même classe partagent les mêmes attributs : si l'on modifie un attribut de l'un, l'autre est modifié aussi.

■

Exercice 0.2.

On exécute le programme suivant, la classe `Eleve` étant celle du cours :

```
eleve = Eleve()
print(eleve.nom)
print(eleve.specialite)
```

Que va afficher ce programme ? Justifier en rappelant le rôle du constructeur. ■

Exercice 0.3.

Dans la classe suivante, lister les attributs, puis les méthodes.

```
class Chien:
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age

    def aboyer(self):
        print("Wouf !")
```

■

■ **Exercice 0.4.**
Compléter le constructeur de la classe `Livre` pour que chaque objet possède les attributs `titre`, `auteur` et `nb_pages`, initialisés avec les valeurs passées en paramètres.

```
class Livre:
    def __init__(self, titre, auteur, nb_pages):
        ...
```

■

■ **Exercice 0.5.**
Parmi les instructions suivantes, lesquelles sont des appels de méthodes ? Rappel : une méthode s'appelle avec un point entre l'objet et le nom de la méthode.

```
"NSI".upper()
len("NSI")
[1, 2, 3].append(4)
abs(-5)
premier_eleve.dire_present()
```

■

Exercices d'application

Les exercices d'application demandent d'écrire des classes complètes, de créer des objets et de faire interagir des objets entre eux.

■ **Exercice 0.6.**
L'objectif de cet exercice est de faire une classe `Professeur` et de créer deux objets : Gorce et Gibaud, avec les bons attributs.

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

■

■ **Exercice 0.7.**
Ajouter à la classe `Professeur` une méthode prenant en argument un texte (qui sera un texte d'exercice) et qui affiche cet exercice. ■

■ **Exercice 0.8.**
On considère la classe `Livre` suivante :

```
class Livre:
    def __init__(self, titre, auteur, nb_pages):
        self.titre = titre
        self.auteur = auteur
        self.nb_pages = nb_pages
```

1. Créer un objet `livre1` représentant le livre « Les Misérables » de Victor Hugo (1462 pages).
2. Ajouter une méthode `resumer` qui affiche une phrase du type : « Les Misérables, écrit par Victor Hugo, 1462 pages ». On pourra utiliser une chaîne formatée (f-string).
3. Afficher l'attribut `titre` de `livre1`.

■

■ **Exercice 0.9.**
On veut modéliser des joueurs d'un jeu vidéo.

1. Faire une classe `Joueur` avec comme attributs un identifiant (qui sera un entier positif ou nul) et un score (qui sera aussi un entier).
2. Ajouter à cette classe `Joueur` une méthode (vous choisirez les arguments de la méthode) qui affiche l'identifiant et le score dans un message lisible.
3. 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 : le score d'un joueur ne peut pas devenir négatif.

```
import random

class Joueur:
    def __init__(self, identifiant, score):
        self.identifiant = identifiant
        self.score = score
```

■

■ **Exercice 0.10.**
On reprend la classe `Joueur` de l'exercice précédent. Faire une classe `Groupe_de_Joueurs` dont l'attribut est une liste de `Joueur`, et qui aura comme méthodes :

1. `trier_la_liste`, qui trie les joueurs par score croissant (on pourra utiliser la fonction `sorted` avec le paramètre `key`, ou écrire un tri par insertion comme en Première) ;

2. `voler_dans_tous_les_sens`, qui prend en argument `n` et qui fait tourner `n` vols entre joueurs du groupe (à chaque vol, on choisit au hasard deux joueurs différents), puis qui trie ensuite la liste des joueurs.

```
class Groupe_de_Joueurs:
    def __init__(self, joueurs):
        self.joueurs = joueurs
```

■

■ **Exercice 0.11.**
Ajouter à la classe `Eleve` du cours une méthode `afficher_fiche` qui affiche le nom, le sexe et la liste des spécialités de l'élève, sous la forme :

```
Isma (Femme)
Specialites : ["Math", "NSI", "HLP"]
```

Tester cette méthode sur un objet de votre choix. ■

Exercices de réflexion

Les exercices de réflexion demandent de concevoir des classes plus complètes et d'analyser des situations limites.

■ **Exercice 0.12.**
On veut modéliser des points du plan.

1. Créer une classe `Point` avec les attributs `x` et `y`, initialisés par le constructeur.
2. Ajouter une méthode `afficher` qui affiche les coordonnées du point sous la forme `(x ; y)`.
3. Ajouter une méthode `distance_origine` qui renvoie la distance du point à l'origine O(0 ; 0) du repère. On rappelle que cette distance vaut $\sqrt{x^2 + y^2}$, et que `math.sqrt` est disponible après `import math`.
4. Créer deux points, afficher leurs coordonnées, puis afficher leur distance à l'origine.

```
import math

class Point:
    ...
```

■

■ **Exercice 0.13.**

On veut modéliser un compte bancaire.

1. Créer une classe `CompteBancaire` avec les attributs `titulaire` et `solde`, initialisés par le constructeur (le solde initial est donné en paramètre).
2. Ajouter une méthode `deposer` qui ajoute une somme au solde.
3. Ajouter une méthode `retirer` qui retire une somme du solde, uniquement si le solde reste positif ou nul : sinon, afficher un message d'erreur et ne pas modifier le solde.
4. Ajouter une méthode `__str__` qui renvoie une chaîne décrivant le compte, par exemple `Compte de Aya : solde 120 euros`.

```
class CompteBancaire:
    ...
```

■

■ **Exercice 0.14.**

On considère le programme suivant, la classe `Eleve` étant celle du cours :

```
eleve_a = Eleve()
eleve_b = Eleve()
eleve_a.nom = "Aya"
print(eleve_b.nom)
```

1. Que va afficher ce programme ? Justifier.
2. Que se passe-t-il si, dans la définition de `dire_present`, on oublie le paramètre `self` ? Expliquer l'erreur obtenue.
3. Peut-on créer deux objets de la même classe avec des attributs différents ? Justifier.

■