Programmation Orientée Objet

A. Introduction :

Jusqu'à présent nous avons vu un seul paradigme de programmation : la **programmation impérative.**
Un paradigme de programmation est une façon d'approcher la programmation informatique et de traiter les solutions aux problèmes et leur formulation dans un langage de programmation approprié
Les programmes que nous avons écrits utilisent une approche **procédurale** :

- On définit les variables qui représentent ce qu'on souhaite modéliser
- On crée les fonctions qui vont changer l'état de ces variables
- L'exécution coordonnée de ces fonctions fait passer nos variables d'un état à l'autre.

Cette approche permet de résoudre tous les problèmes informatiques.
Néanmoins elle présente un défaut majeur : il est difficile de dégager une structure à notre code.
Cela le rend peu lisible et difficile à entretenir. Autre inconvénient, nos programmes ne sont pas ré-exploitables facilement. Pour réutiliser une fonction déjà écrite, la seule approche efficace est, pour l'instant, d'en faire un copier-coller.

Un nouveau paradigme de programmation :

La programmation orientée objet (POO), qui fait ses débuts dans les années 1960 avec les réalisations dans le langage Lisp, a été formellement définie avec les langages Simula (vers 1970) puis SmallTalk. Puis elle s'est développée dans les langages anciens comme le Fortran, le Cobol et est même incontournable dans des langages récents comme Java.

La **programmation orientée objet** (POO) est un paradigme de programmation informatique. 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. Un **objet** se caractérise par son **état**, son **comportement**, son identité

- L'état est défini par les valeurs des **attributs** de l'objet à un instant t.
  Par exemple, pour un téléphone, certains attributs sont variables dans le temps comme allumé ou éteint, d'autres sont invariants comme le modèle de téléphone.
- Le comportement est défini par les **méthodes** de l'objet : en résumé, les méthodes définissent à quoi sert l'objet et/ou permettent de modifier son état.
- L'identité est définie à la déclaration de l'objet (instanciation) par le nom choisi, tout simplement.

B. Les Classes

En **programmation orientée objet**, les différents objets utilisés peuvent être construit indépendamment les uns des autres (par exemple par des programmeurs différents) sans qu'il y ait de risque d'interférence.
Ce résultat est obtenu grâce au concept d'**encapsulation** : la fonctionnalité interne de l'objet et les variables qu'il utilise pour effectuer son travail, sont en quelques sortes enfermées dans l'objet.
Les autres objets et le monde extérieur ne peuvent y accéder qu'à travers des procédures bien définies, c'est ce qu'on appelle **l'interface** de l'objet.
Une **classe** est une description d'un ensemble d'objets ayant une structure de données communes (**attributs**) et pouvant réaliser des actions (**méthodes**). On considère en fait une classe comme un nouveau type de données. On appelle **instance** de la classe l'objet qui la représente.

C. 1er exemple de classe en Python:

```
1   class Combattant:
2       nb_combattants=0
3       def __init__(self,vie,attaque):
4           """ création de l'instance avec ses attributs'
5           self.vie=vie
6           self.attaque=attaque
7           self.vivant=True
8           Combattant.nb_combattants+=1
9
10      def perdre_vie(self,points):
11          """ enlève des points de vie et peut modifier
12              l'état du combattant"""
13          self.vie=self.vie-points
14          if self.vie < 0:
15              self.vivant=False
16
17      def nb_vies(self):
18          """ retourne le nb de vies de l'instance"""
19          return self.vie
20
21      def etat(self):
22          """ retourne l'état de l'instance"""
23          return self.vivant
24
25      def attaquer(self,adversaire):
26          """ fait des perdre des points de vie à
27              l'instance autre"""
28          adversaire.perdre_vie(self.attaque)
```

Commentaires sur le code de la classe.

- Ligne 1. On utilise le mot clé class suivi du nom de la classe écrit avec une **majuscule.**
- Lignes 2 à 28. La définition de la classe se termine quand on revient à l'indentation.
- Ligne 2 : On définit si besoin un **attribut de classe**
- Ligne 3, 10,17,21,25 : définition des *méthodes (fonctions de la classe)*
  La variable **self** est toujours le premier paramètre d'une méthode. Il désigne l'objet lui- même.
- Ligne 3 à 8 : La méthode **__init__** est appelée "constructeur" qui crée un objet du type Combattant
- Ligne 5,6,7 : On crée les **attributs d'instance,** on utilise self.attribut pour accéder à un attribut.
- Ligne 8 : l'attribut de classe est appelé par nom_classe.nom_attribut et incrémenté dans la méthode
- Ligne 25 à 28 : cette méthode fait appel à une autre méthode de la classe

A retenir

- La variable **self**, dans les méthodes d'un objet, désigne l'objet auquel s'appliquera la méthode.
  Elle représente l'objet dans la méthode en attendant qu'il soit créé.
  Attention ! Python n'impose pas le nom self. . . ce n'est qu'un *usage, mais tout le monde l'utilise !*
- **Les attributs :** Les attributs sont des variables associées à la classe. Il y en a de deux types :
  - **Les attributs de classe** : un attribut de classe est un attribut qui sera identique pour chaque instance et n'a pas vocation à être changé.
  - **Les attributs d'instance** : Une variable ou attribut d'instance est une variable accrochée à une instance et qui est spécifique à cette instance

Remarques :

- L'attribut de classe **nb_combattants** n'est pas propre à une instance mais bien à toute la classe.
- La méthode attaquer prend en paramètre un objet **adversaire** qui est lui-même un objet du type Combattant.

Utilisation de la classe Combattant :

La classe est une espèce de moule et à partir de ce moule nous allons créer des objets (plus exactement nous parlerons **d'instances**). Pour **créer** une instance, la procédure est relativement simple.
Ici, on crée 2 instances guerrier et magicien avec 2 attributs (vie et attaque), l'attribut vivant est par défaut initialisé à True.

Pour **accéder** aux valeurs des attributs d'une instance, l'instruction est **nom_instance.nom_attribut** ou par une méthode crée dans la classe **nom_instance.nom_methode()**

Pour **appliquer** une méthode à une instance , l'instruction est nom_instance.nom_méthode () avec des éventuels paramètres entre parenthèses .
Attention : self n'est pas à passer dans les paramètres.

```
In [6]: guerrier=Combattant(10,4)

In [7]: type(guerrier)
Out[7]: __main__.Combattant

In [8]: magicien=Combattant(20,2)

In [9]: magicien.vie
Out[9]: 20

In [10]: magicien.nb_vies()
Out[10]: 20

In [11]: magicien.perdre_vie(15)

In [12]: magicien.nb_vies()
Out[12]: 5

In [13]: guerrier.attaquer(magicien)

In [14]: magicien.nb_vies()
Out[14]: 1

In [15]: guerrier.attaquer(magicien)

In [16]: magicien.etat()
Out[16]: False
```