PGR. Programmation orientée objet (bases)

I/ Introduction

La POO est un **paradigme** de programmation, au même titre que la programmation **impérative** (vue en première) ou la programmation **fonctionnelle** (qui sera étudiée cette année en Terminale), ou encore d'autres paradigmes (événementielle par exemple).
Un **paradigme de programmation** pourrait se définir comme une *philosophie* dans la manière de programmer : c'est un parti-pris revendiqué dans la manière d'aborder le problème à résoudre. Une fois cette décision prise, des outils spécifiques au paradigme choisi sont utilisés.

Exemple : On peut imaginer deux menuisiers qui ont pour mission de fabriquer chacun un meuble.

- Le premier pourra décider d'utiliser du collé-pointé : il assemblera les morceaux de bois en les collant puis utilisera des pointes. Ses outils seront le marteau et le pistolet à colle.
- Le deuxième pourra décider de visser les morceaux de bois entre eux : son outil principal sera une visseuse.

Pour la réalisation de sa mission, chaque menuisier utilise un paradigme différent. Qui utilise la meilleure méthode ? Cette question n'a pas vraiment de réponse : certaines méthodes sont plus rapides que d'autres, d'autres plus robustes, d'autres plus esthétiques...
Et pourquoi ne pas mélanger les paradigmes ? Rien n'interdit d'utiliser des pointes ET des vis dans la fabrication d'un meuble.

La Programmation Orientée Objet *(notée **POO** en abrégé*) est souvent mélangée avec de la programmation impérative, de la programmation fonctionnelle etc.

II/ Python, un langage résolument orienté objet

Voici un petit exemple d'un programme en Python autour d'une *list*.

Déclaration d'une liste

```
m = [4,5,2]
```

```
type(m)
```

```
list
```

Inversion de la liste

```
m.reverse()
```

```
print(m)
```

```
[2, 5, 4]
```

Lors de la déclaration de la liste, on voit que le *type* de la variable *m* est une liste. Cela signifie qu'elle fait partie de la **classe** *list*, on parlera **d'instance** en POO : *m* est alors un **objet** de la classe *list*.

Lors de l'inversion de la liste, on constate que *m.reverse()* permet d'inverser l'ordre des éléments de la liste. La fonction *reverse()* est appelée **méthode** (ou ***fonction-membre*** en langage C++) car elle ne peut être utilisée que pour les instances de la classe *list* (sauf si évidemment d'autres classes possèdent cette méthode).

L'utilisateur n'a pas besoin de savoir « comment » la méthode *reverse()* fonctionne : il s'agit du principe d'**encapsulation**.

Le point séparant l'objet *m* de la méthode *reverse()* est ce qui indique au langage Python l'appel à cette méthode. On retrouve d'ailleurs cette notation dans beaucoup de langages orientés objet (une flèche en C++ s'il s'agit d'un pointeur explicite).

On remarquera ici la **grande utilité de donner des noms explicites aux méthodes** : ici, le rôle de la méthode *reverse()* est relativement évident !

On trouvera ici l'ensemble des méthodes de la classe *list* :
https://docs.python.org/fr/3/tutorial/datastructures.html#using-lists-as-stacks

En savoir plus : Le langage Python et la programmation orientée objet, https://www.pierre-giraud.com/python-apprendre-programmer-cours/introduction-oriente-objet/

III/ Créer sa propre classe

1/ Vocabulaire

Jusqu'ici on a employé uniquement le mot «objet». Il convient maintenant d'être plus précis.
On désignera par **classe** la structure de données définissant une catégorie générique d'objets. Dans le monde animal, *chat* est une classe (nommée en réalité *félidé*).

Chaque élément de la classe *chat* va se distinguer par des caractéristiques : un âge, une couleur de pelage, un surnom... (on appellera ces caractéristiques des **attributs** ou ***données-membres*** en C++) et des fonctionnalités, comme la **méthode** *attrape_souris()*. Lorsqu'on désigne un chat en particulier, on désigne alors un **objet** (bien réel) qui est une **instance** de la **classe** (abstraite) *chat*.

Par exemple, l'**objet** *larry* est une **instance** de la **classe** *chat* . Il a un pelage et un surnom particulier.

```
larry.pelage = "blanc et tabby"
larry.surnom = "Chief Mouser to the Cabinet Office"
```

2/ La méthode « constructeur »

Même si le langage Python est très permissif à ce niveau, il convient d'indiquer dès le début tous les attributs que l'on souhaite créer. Pour cela, le **constructeur** est la bonne technique à utiliser. Comme son nom l'indique, le constructeur va permettre d'instancier des objets d'une classe. Il ne restera plus qu'à créer les méthodes via le mot clé habituel *def*.

Voici un exemple :

```
class Voiture :
    def __init__(self, annee, coul, vmax) :
        self.annee = annee
        self.couleur = coul
        self.vitesse_max = vmax
        self.age = 2020 - self.annee
```

Le mot clé ***self*** permet de cibler l'objet en question (il est souvent appelé *this* notamment en C++ et Java). La **méthode __init__(self, paramètres ..)** permet la création de l'objet et de ses attributs : elle est donc obligatoire dans toute création de classe. Les *underscore* autour de *init* signalent qu'il s'agit d'une méthode dite privée, c'est-à-dire

utilisée dans toute classe quel qu'elle soit : en général, l'utilisateur n'a pas à les appeler sauf dans quelques cas comme celui-ci -et la méthode *__str__()* (si besoin) qui permet « d'écrire » un objet à l'écran par exemple-.

Important : On notera également que le mot clé *self* doit apparaître dans tout paramètre d'une méthode. Il y aura une erreur d'exécution le cas échéant, notamment sur le nombre d'arguments appelé !

A noter (ce n'est pas à savoir par cœur) :

- Un paramètre est un **type** de variable : c'est ce qui apparaît entre les parenthèses d'une fonction (ou méthode). Cela fait partie de l'aspect déclaratif de fonctions (langages C / C++ notamment).
- Un argument est la **valeur** passée lorsque l'on appelle une fonction (ou méthode).

Comment peut-on ajouter des méthodes dans une classe ? Voici un exemple :

```
class Voiture :
    def __init__(self, annee, coul, vmax) :
        self.annee = annee
        self.couleur = coul
        self.vitesse_max = vmax
        self.age = 2020 - self.annee

    def petite_annonce(self) :
        print("À vendre voiture", self.couleur, "de", self.annee, ", vitesse maximale", self.vitesse_max, "km/h.")
```

```
batmobile = Voiture(2036, "noire", 325)
```

```
batmobile.petite_annonce()
```

```
À vendre voiture noire de 2036 , vitesse maximale 325 km/h.
```

Ici, la méthode *petite_annonce(self)* permet « d'afficher l'objet ». On remarquera plusieurs choses :

- **L'indentation** au niveau des méthodes : Python sait qu'il s'agit donc de méthodes et non de fonctions classiques.
- **Le mot clé *self*** dans les paramètres de la méthode *petite_annonce(self)*. En revanche, il ne faut pas le spécifier lorsqu'un objet l'appelle : cela est logique, le « point » entre *l'instance* et la *méthode* supprime tout éventuelle ambiguïté.

On notera que seules les instances de la classe *Voiture* peuvent appeler la méthode *petite_annonce()*.

Aller plus loin en POO en Python (3 vidéos qui se suivent)
https://www.youtube.com/watch?v=91dPooHyNIo
Auteur : FormationVidéo, durée : 25 min 27 sec
https://www.youtube.com/watch?v=B-OlnwpVBN4
Auteur : FormationVidéo, durée : 19 min 05 sec
https://www.youtube.com/watch?v=Fs6XsN6masA
Auteur : FormationVidéo, durée : 23 min 45 sec