Chapitre 3 - Programmation Orientée Objet (POO)

I - Introduction : Programmation orientée objet

1) La notion d'objet et de classe

Jusqu'ici, les programmes ont été réalisés en programmation procédurales, c'est à dire que chaque programme a été décomposé en plusieurs fonctions réalisant des tâches simples.
Cependant lorsque plusieurs programmeurs travaillent simultanément sur un projet, il est nécessaire de programmer autrement afin d'éviter les conflits entre les fonctions.

Un **objet** se caractérise par 3 choses :

- 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.

En programmation orientée objet, on fabrique de nouveau types de données correspondant aux besoin du programme. On réfléchit alors aux caractéristiques des objets qui seront de ce type et aux actions possibles à partir de ces objets.
Ces caractéristiques et ces actions sont regroupées dans un code spécifique associé au type de données, appelé **classe**.

2) Classe : un premier exemple avec le type list

Le type de données *list* est une classe .

```
# Dans la console PYTHON
>>> l=[1,5,2]
>>> type(l)
<class 'list'>
```

Une action possible sur les objets de type liste est le tri de celle-ci avec la méthode nommée sort().
On parle alors de **méthode** et la syntaxe est : $nom\_objet.nom\_methode$
Exemple avec la classe list : *liste.sort()*

```
# Dans la console PYTHON
>>> l=[1,5,2]
>>> l.sort()
>>>l
[1,2,5]
```

3) Vocabulaire

- Le type de données avec ses **caractéristiques** et ses **actions** possibles s'appelle **classe**.
- Les **caractéristiques** (ou **variables**) de la classe s'appellent les **attributs**.
- Les **actions** possibles à effectuer avec la classe s'appellent les **méthodes**.

  La classe définit donc les attributs et les actions possibles sur ces attributs, les méthodes.

- Un objet du type de la classe s'appelle une **instance** de la classe et la création d'un objet d'une classe s'appelle une **instanciation** de cette classe.
- On dit que les attributs et les méthodes sont **encapsulés** dans la classe. On parle aussi d'**encapsulation**.

On peut afficher les méthodes associées à un objet avec la fonction dir(objet) :

```
# Dans la console PYTHON
>>> l=[1,5,2]
>>> dir(l)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__dir__', '__doc__','__eq__', '__format__', '__ge__', '__getattribute__', '
   __getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '
   __init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__reversed__', '__rmul__', '__setattr__','__setitem__', '__sizeof__',
'__str__', '__subclasshook__', 'append', 'clear', 'copy','count',
'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

On retrouve les méthodes connues concernant les listes : **sort(), count(), append(), pop()** ...
Les autres méthodes encadrées par les underscores sont spéciales.

4) Un peu d'histoire

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

II - Création d'une classe pas à pas

1) Un constructeur

- On va créer une classe simple, la classe **Carte** correspondant à une carte d'un jeu de 32 ou 52 cartes.
  Par convention, une classe s'écrit toujours avec une majuscule.

```
# Dans l'editeur PYTHON
class Carte:
        "Une carte d'un jeu de 32 ou 52 cartes"
```

- Une **méthode constructeur** commence toujours par :

```
def __init__(self,...):
```

  Le paramètre particulier self est expliqué en fin de chapitre 2.1. Avec deux tirets bas ou underscores (AltGr 8) de part et d'autre de init.

- Dans le constructeur de la classe (méthode __init__), on définit les **attributs** de la **classe Carte** qui seront :
  (a) sa valeur 2, 3 $\cdots$, 10, 11 pour Valet, 12 pour Dame, 13 pour Roi et 14 pour As ;
  (b) sa couleur (Carreau, Coeur, Trèfle, Pique).

```
class Carte:
        # Definition de la classe
        "Une carte d'un jeu de 32 ou 52 cartes"
        def __init__(self,valeur,couleur):
                # constructeur
                # 1er attribut valeur {de 2 a 14 pour as}
                self.valeur=valeur
                # 2e attribut {'pique','carreau','coeur','trefle'}
                self.couleur=couleur
```

2) Instanciation d'un objet

- **Création d'une instance**
  Dans un programme ou fonction (en dehors de la classe), la création d'une **instance** de la classe Carte se fait de la manière suivante :

```
# Dans un programme ou une fonction PYTHON
x=Carte(5,'carreau')
```

  La variable x est un objet de la classe Carte dont les attributs sont $valeur = 5$ et $couleur = carreau$.
  Par abus de langage, on peut dire qu'une classe définie un nouveau type de donnée, ici le type de donnée "Carte".
  Lorsque l'on créé un objet, son constructeur est appelé implicitement et l'ordinateur alloue de la mémoire pour l'objet et ses attributs.
  On peut d'ailleurs obtenir l'adresse mémoire de notre objet créé x.

```
# Dans un programme ou une fonction PYTHON
x=Carte(5,'carreau')
print(x)
#Output
<__main__.Carte object at 0x7f7f57d4ae90>
```

L'affichage indique ici que x est un objet de la classe Carte ('Carte object') et son adresse mémoire est 0x7f7f57d4ae90.

- **Obtention de la valeur d'un attribut**
  Pour obtenir la valeur d'un attribut d'un objet, il faut utiliser l'opérateur d'accessibilité . (point) de la façon suivante : ***nom_objet.nom_attribut***.
  Cela peut se lire ainsi de droite à gauche nom_attribut appartenant à l'instance nom_objet.

```
# Dans un programme ou une fonction PYTHON
x=Carte(5,'carreau')
v = x.valeur
c = x.couleur
print("Couleur : ",c," Valeur : ",v)
#Ouput : Couleur : 5 Valeur : carreau
```

- **La variable self**
  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.

```
# Dans le fichier Python de la classe Carte
class Carte:
        # Definition de la classe
        "Une carte d'un jeu de 32 ou 52 cartes"
        def __init__(self,valeur,couleur): # constructeur
                self.valeur=valeur
                # 1er attribut
                self.couleur=couleur # 2e attribut
```

```
# Dans un programme ou fonction python (en dehors de la classe Carte)
x=Carte(5,'carreau')
y=Carte(14,'pique')
```

Dans cet exemple, la méthode __init__ (constructeur) est appelée implicitement. **"self"** fait référence à l'objet x dans la première ligne et à l'objet y dans la seconde.

3) Encapsulation : les accesseurs (ou ”getters”)

On ne va généralement pas utiliser la méthode précédente nom_objet.nom_attribut permettant d'accéder aux valeurs des attributs car on ne veut pas forcement que l'utilisateur ait accès à la représentation interne des classes.
Pour utiliser ou modifier les attributs, on utilisera de préférence des méthodes dédiées dont le rôle est de faire l'**interface** entre l'utilisateur de l'objet et la représentation interne de l'objet (ses attributs).
Les attributs sont alors en quelque sorte **encapsulés** dans l'objet, c'est à dire non accessibles directement par le programmeur qui a instancié un objet de cette classe.

- L'**encapsulation** désigne le principe de regrouper des données brutes avec un ensemble de routines (méthodes) permettant de les lire ou de les manipuler.
- But de l'encapsulation : cacher la représentation interne des classes.
  (a) pour simplifier la vie du programmeur qui les utilise ;
  (b) pour masquer leur complexité (diviser pour régner) ;
  (c) pour permettre de modifier celle-ci sans changer le reste du programme.
  (d) la liste des méthodes devient une sorte de mode d'emploi de la classe.

Pour obtenir la valeur d'un attribut nous utiliserons la méthode des **accesseurs** (ou ”getters”) dont le nom est généralement : ***getNom_attribut()*** . Exemple :

```
# Dans le fichier Python de la classe Carte
class Carte:
        # Definition de la classe
        "Une carte d'un jeu de 32 ou 52 cartes"
        def __init__(self,valeur,couleur):
                # methode 1 : constructeur
                # 1er attribut valeur {de 2 a 14 pour as}
                self.valeur=valeur
                # 2e attribut {'pique','carreau','coeur','trefle'}
                self.couleur=couleur

        def getAttributs(self):
                # methode 2 : permet d'acceder aux valeurs des attributs
                return (self.valeur,self.couleur)
```

```
# Dans un programme ou fonction python (en dehors de la classe Carte)
x=Carte(5,'carreau')
y=Carte(14,'pique')
print(x)
#Output : <__main__.Carte object at 0x7f7f57d4af50>
print(y)
#Output : <__main__.Carte object at 0x7f7f57d4ae90>
print(x.getAttributs())
#Output : (5,'carreau')
print(y.getAttributs())
#Output : (14,'pique')
```

Explication du programme :

1. *x=Carte(5,'carreau')* : On appelle le constructeur __init__() de la classe pour instancier l'objet x.
2. *y=Carte(14,'pique')* : On appelle le constructeur __init__() de la classe pour instancier l'objet y.
3. *print(x)* : On affiche l'adresse mémoire de l'objet x.
4. *print(y)* : On affiche l'adresse mémoire de l'objet y.
   s Les deux adresses sont différentes et **pointent** donc vers deux objets différents.
5. *print(x.getAttributs())* : On appelle la méthode *getAttributs* de la classe Carte. Cette méthode est appliquée à l'objet x et retourne les attributs 'valeur' et 'couleur' de l'objet x. La fonction *print* affiche les attributs.
6. *print(y.getAttributs())* : On appelle la méthode *getAttributs* de la classe Carte. Cette méthode est appliquée à l'objet y et retourne les attributs 'valeur' et 'couleur' de l'objet y. La fonction *print* affiche les attributs.

4) Modifications contrôlées des valeurs des attributs : les mutateurs (ou "setters")

On souhaite contrôler les valeurs attribuées aux attributs. Pour cela, on passe par des méthodes particulières appelées mutateurs (ou "setters") qui vont modifier la valeur d'une propriété d'un objet. Le nom d'un mutateur est généralement : ***setNom_attribut()*** .

```
# Dans le fichier Python de la classe Carte
class Carte: # Definition de la classe
        "Une carte d'un jeu de 32 ou 52 cartes"
        def __init__(self,valeur,couleur):
                # methode 1 : constructeur
                # 1er attribut valeur {de 2 a 14 pour as}
                self.valeur=valeur
                # 2e attribut {'pique','carreau','coeur','trefle'}
                self.couleur=couleur

        def getAttributs(self):
                # methode 2 : permet d'acceder aux valeurs des attributs
                return (self.valeur,self.couleur)

        def getValeur(self):
                # methode 3 : accesseur
                return self.valeur

        def getCouleur(self):  # methode 4 : accesseur
                return self.valeur

        def setValeur(self,v): # methode 5 : mutateur avec controle
                if 2<=v<=14:
                        self.valeur=v
                        return True
                else:
                        return False
```

Exemple d'utilisation :
Création une carte c1, un 7 de coeur puis modification de sa valeur en la passant à 10.

```
# Dans un programme ou fonction python (en dehors de la classe Carte)
c1=Carte(7,'coeur')
print(c1.getAttributs())
#Output : (7, 'coeur')
c1.setValeur(10)
print(c1.getAttributs())
#Output : (10, 'coeur')
```

III - BILAN

Classe, attributs, méthodes, accesseurs et mutateurs

- Le type de données avec ses caractéristiques et ses actions possibles s'appelle **classe**.
- Les caractéristiques (ou variables) de la classe s'appellent les **attributs**.
- Les actions possibles à effectuer avec la classe s'appellent les **méthodes**.
- La classe définit donc les attributs et les actions possibles sur ces attributs, les méthodes.
- **Constructeur** : la manière « normale » de spécifier l'initialisation d'un objet est d'écrire un constructeur .
- L'**encapsulation** désigne le principe de regrouper des données brutes avec un ensemble de routines (méthodes) permettant de les lire ou de les manipuler.
- **Accesseur** ou « getter » : une fonction qui retourne la valeur d'un attribut de l'objet. Par convention son nom est généralement sous la forme : *getNom_attribut()*.
- **Mutateur** ou setter : une procédure qui permet de modifier la valeur d'un attribut d'un objet. Son nom est généralement sous la forme : *setNom_attribut()*.