Première NSI

Chapitre I - Classes et programmation objet

Nous avons vu en classe de première les paradigmes de programmation impératif (séquence d'instruction les unes à la suite des autres). Cette année, nous allons voir un nouveau paradigme de programmation : le paradigme objet, ou programmation objet.
Un **type** de données définit :

- l'ensemble des valeurs possibles pour les données de ce type
- les opérations applicables sur ces données

Deux exemples

- Les *booléens* :
  - L'ensemble des valeurs est : True, False
  - Des opérations possibles : not, and, or, etc.
- Les *entiers* :
  - L'ensemble des valeurs : 0, 1, 2, 3, 4, ..., -1, -2, -3, -4, ...
  - Des opérations possibles : $+$, $-$, $*$, $//$.

Une **classe** définit la structure d'un **objet**. Elle permet donc de créer de nouvelles structures de données (et donc de nouveaux types de données).
On appelle **instance** d'une classe un objet créé par une classe.
La classe définit :

- la liste des **attributs**, c'est-à-dire la liste des données qui appartiennent à l'objet. Ils définissent les **états** de l'objet.
- la liste des **méthodes**, c'est-à-dire la liste des fonctions qui appartiennent à cette classe et qui ne s'appliquent que sur des objets de cette classe. Cela définit le comportement et les actions de l'objet.

Exemple de classe en python

```
class Colis():
    valeur=7
    def __init__(self,masse,adresse):
        """
        constructeur: initialise les états de l'objet de type Colis
        :param self: (Colis) un objet de type Colis
        :param masse: (float) un nombre flottant
        :param adresse: (string) une chaîne de caractères
        :return: (Nonetype)
        """
        self.__masse = masse
        self.__adresse = adresse
    def donne_adresse(self):
        """
        sélecteur : renvoie l'adresse de l'objet de type Colis
        :param self: (Colis) un objet de type Colis
        :return: (string) une chaîne de caractères
        """
        return self.__adresse
    def donne_masse(self):
        """
        sélecteur : renvoie la masse de l'objet de type Colis
        :param self: (Colis) un objet de type Colis
        :return: (float) un nombre flottant
        """
        return self.__masse
```

Exemple - suite

```
27      def attribue_nouvelle_adresse(self, adresse):
28          """
29          attribue une nouvelle adresse à l'objet de type Colis
30          :param self: (Colis) un objet de type Colis
31          :param adresse: (string) une chaîne de caractères
32          :return: (Nonetype)
33          """
34          self.__adresse = adresse
```

Les objets de la classe `Colis` possèdent 3 attributs (`__masse` , `__destination` et `valeur`) et 4 méthodes (`__init__()` , `donne_adresse()` , `donne_masse()` et `attribue_nouvelle_adresse()`).

Remarque

- Toutes les méthode ont au moins un paramètre : `self` qui fait référence à l'objet sur lequel on utilise la méthode. Cependant, cet objet est implicite lorsqu'on utilise la méthode et n'est donc pas à fournir en paramètre.
- Une méthode s'appelle `__init__()` . Elle est obligatoire. C'est elle qui initialise l'objet. Cette méthode est appelée `constructeur` . Dans notre cas, elle possède 3 paramètres : `self` dont on a parlé précédemment et `masse` et `adresse` qu'il faudra fournir.
- Le nom de certains attributs sont précédés de `__` : ce sont des attributs privés. L'utilisateur ne peut pas y accéder. D'autres n'ont pas de `__` : ils sont publics. L'utilisateur peut y accéder. Les valeurs données aux attributs dans la classe, hors des méthodes permettent d'initialiser ces attributs.

```
1  >>> mon_colis = Colis(3.4, 'chez toi')
2  >>> mon_colis.donne_adresse()
3  'chez toi'
4  >>> mon_colis.donne_masse()
5  3.4
6  >>> mon_colis.attribue_nouvelle_adresse('chez lui')
7  >>> mon_colis.donne_adresse()
8  'chez lui'
9  >>> mon_colis.valeur
10 7
11 >>> mon_colis.__masse
12 Traceback (most recent call last):
13 .
14 .
15 .
16 AttributeError: 'Colis' object has no attribute '__masse'
```