1 L'approche objet en programmation

En programmation, l'approche objet consiste à penser le système d'information, ou encore le modèle du monde, sous la forme d'un ensemble d'entités. Chaque entité a ses propres caractéristiques et ses propres fonctionnalités. Chaque entité est maîtresse de son évolution au cours du temps.

Dans le TP Bibliothèque, vous avez manipulé trois types d'entités : les usagers, les livres, et les emprunts.

2 Ma première classe

Prenons l'exemple des usagers. Chaque usager est défini par les mêmes informations (nom, prénom, date de naissance, liste des emprunts), mais chaque usager a ses propres valeurs pour ces informations. En plus de ces caractéristiques, un usager est détenteur de plusieurs fonctionnalités comme "emprunter un livre".

Nous allons définir une classe des usagers, qui va déterminer ce qu'est un usager type, une sorte de squelette d'usager, une coquille vide. Il faudra aussi définir comment à partir de cette coquille vide, on construit un usager réel.

Voici votre première classe :

```
class Usager:
    def __init__(self,nom,prenom,naissance):
        self.nom = nom
        self.prenom = prenom
        self.date_naissance = naissance
        self.emprunts = []
```

3 Les attributs d'un objet

Le code Python ci-dessus définit la classe `Usager`. Cette définition précise qu'un usager est constitué d'un nom (`nom`), d'un prénom (`prenom`), d'une date de naissance (`date_naissance`) et de la liste de ses emprunts (`emprunts`). On appellera **attribut** chacune de ces caractéristiques. Donc, `nom`, `prenom`, `date_naissance`, `emprunts` sont les attributs de la classe `Usager`.

4 Le constructeur, et les composants d'une classe

La classe `Usager` contient une méthode, qui s'appelle `__init__`. Cette méthode construit un objet de la classe. Tout classe doit contenir une telle méthode. Le terme **méthode** fait partie du vocabulaire de l'approche objet :

— Une **classe** est constituée d'**attributs** et de **méthodes**.

— Chaque individu représentant de la classe s'appelle un **objet**. On dit qu'il s'agit d'une **instance** de la classe
— Pour construire un objet de la classe, il faut un **constructeur**. En Python, il s'agit de la méthode `__init__`.

La méthode `__init__` prend en argument au moins `self`, qui fait référence à l'objet en cours de construction. Ici, la ligne `self.nom = nom` se lit : "nom est un attribut de self, et sa valeur initiale est donnée par le contenu de nom, argument de `__init__`. Comme on le remarque avec la ligne `self.date_naissance = naissance`, le nom de l'attribut et celui de l'argument donnant la valeur initiale ne doivent pas nécessairement être les mêmes, mais c'est mieux ainsi pour la lisibilité du code.

5 Construire un objet

Ici, nous n'avons fait que définir **comment** construire un objet de la classe `Usager`. Reste encore à le construire. Voici le code permettant cela :

```
u1 = Usager("Nonyme","Albert","17/09/2000")
```

`u1` est une variable. Pour construire un objet de la classe `Usager`, on utilise le nom de la classe comme une fonction en passant des valeurs en suivant la signature (liste des arguments) de la méthode `__init__`. Attention, en Python, cette dernière phrase est fausse. En effet, la méthode `__init__` de la classe `Usager` liste 4 arguments (`self`, `nom`, `prenom`, `naissance`), mais l'appel n'en donne que 3 (`nom`, `prenom`, `naissance`). En fait, le `self`, à l'appel du constructeur, est implicite : au moment de la construction, il est ajouté, et fait référence à l'objet en cours de construction.

6 Manipuler un objet

Maintenant, l'objet `u1` est construit, et on peut l'utiliser :

```
print(u1.nom)
u1.nom = "Toto"
print(u1.nom)
u1.emprunts.append("23")
print(u1.emprunts)
```

L'opérateur `.` permet d'accéder à un attribut de l'objet `u1`. Cet opérateur peut être "chainée" ; en effet, remarquez l'usage `u1.emprunts.append("23")` : le premier `.` permet d'accéder à la liste des emprunts de `u1`. Cette liste est un objet de type `List`, qui dispose d'une méthode `append`. Le deuxième `.` permet d'appliquer `apprend` à cette liste.

Finalement, le code total est le suivant :

```
# coding utf-8

class Usager:
    def __init__(self,nom,prenom,naissance):
        self.nom = nom
        self.prenom = prenom
        self.date_naissance = naissance
        self.emprunts = []

if __name__ == '__main__':
    u1 = Usager("Nonyme","Albert","17/09/2000")
    print(u1.nom)
    u1.nom = "Toto"
    print(u1.nom)
    u1.emprunts.append("23")
    print(u1.emprunts)
```

Exercice 1 : créez un fichier nommé `usager.tex`, et reprenez le code ci-dessus. Exécutez-le pour vérifier qu'on obtient le résultat attendu.

Exercice 2 : ajoutez à la classe `Usager` l'attribut `date_renouvellement`, qui est une date qui définit quand l'abonnement de l'usager se termine. Mettez à jour le constructeur (ajout d'un nouvel argument) ainsi que l'appel dans le test au bas du fichier. Testez en affichant la nouvelle information.

7 Ajout de méthode

Pour le moment, la classe `Usager` ne contient qu'une seule méthode : `__init__`. Nous allons maintenant ajouter une autre méthode :

```
class Usager:
    ...

    def age(self):
        """ retour l'age de l'usager """
        aujourdhui = datetime.now()
        dn = datetime.strptime(self.date_naissance, '%d/%m/%Y')
        retour = aujourdhui.year - dn.year
        ## on enlève 1 an si la date actuelle est antérieure à
        ## celle de l'anniversaire
        if aujourdhui.month < dn.month:
            retour -= 1
        else:
            if aujourdhui.month == dn.month and aujourdhui.day < dn.day:
                retour -= 1
```

```
        return retour
```

La fonction `age` est définie au sein de la classe `Usager`, comme l'indentation devrait vous le faire comprendre. Cette fonction ne prend qu'un seul argument, `self`, nécessaire afin qu'on puisse se référer à l'usager sur laquelle la fonction s'applique. `dn` et `aujourdhui` sont des objets de type `date` qui disposent des attributs `year`, `month`, et `day`. On ne commente pas plus ici l'algorithme de calcul de l'âge.

A noter l'appel à la fonction `now` (`datetime.now()`), qui peut apporter de la confusion ici. En effet, en Python, l'opérateur `.` est aussi utilisé pour accéder à une fonction d'une librairie (ici, la librairie `datetime`).

Exercice 3 : ajoutez à la classe `Usager` une méthode nommée `renouvellement` qui renvoie `True` si la date de renouvellement de l'usager est dépassée, `False`, sinon.