M1102: Introduction à l'algorithmique et à la programmation

Feuille de TD n°11

Les objets en Python

INSTRUCTIONS: à la fin du TD, vous devez déposer sous Celene une archive zip contenant les fichiers Python correspondant à ce TD (fournis avec la feuille). Vous pourrez déposer une version améliorée de votre travail jusqu'à la **VEILLE** du TD suivant.

L'évaluation de la période 2 sera basée sur ces rendus.

Objectifs

L'objectif de cette feuille est d'introduire la programmation orientée objet en Python en montrant comment on passe d'une API comme celles vues dans les feuilles précédentes aux objets.
L'idée principale par laquelle on va introduire les objets repose sur l'intégration de l'API dans la structure de données. Lorsque que l'on a défini des API dans les feuilles précédentes, on a d'un coté l'interface de programmation et de l'autre les variables qui contiennent la structure de données. Par exemple pour les matrices, on a d'un coté la liste des fonctions qui manipulent les matrices et de l'autre on a des variables qui contiennent la structure de données choisie pour représenter les matrices.
La programmation orientée objet va intégrer l'interface de programmation à l'intérieur de la structure de données. De cette manière, les variables (objets) contiendront non seulement la structure de données mais aussi les opérations que l'on peut appliquer dessus.
Voici concrètement sur l'exemple des matrices comment cela se traduit.

```
#----------------------------------------
# Une implémentation des matrices 2D en Python
#----------------------------------------
def Matrice(nbLignes,nbColonnes,valeurParDefaut=0):
    return {"nbLignes":nbLignes,"nbColonnes":nbColonnes,"lesValeurs"
       :[valeurParDefaut]*(nbLignes*nbColonnes)}

def getNbLignes(matrice):
    return matrice["nbLignes"]

def getNbColonnes(matrice):
    return matrice["nbColonnes"]

def getVal(matrice,lig,col):
    return matrice["lesValeurs"][lig*matrice["nbColonnes"]+col]

def setVal(matrice,lig,col,val):
    matrice["lesValeurs"][lig*matrice["nbColonnes"]+col]=val
```

Dans cet exemple on a d'un coté la structure de données qui est un dictionnaire et de l'autre les fonctions de l'API qui permettent de manipuler les matrices. On peut noter qu'à part la première, toutes les fonctions prennent la matrice que l'on veut manipuler comme premier paramètre. Voici maintenant la version orientée objet :

```python
class Matrice(object):

    def __init__(self,nbLignes,nbColonnes,valeurParDefaut=0):
        """
        constructeur
        """
        self._nbLignes=nbLignes
        self._nbColonnes=nbColonnes
        self._listeDesValeurs=[valeurParDefaut]*(nbLignes*nbColonnes)

    def getNbLignes(self):
        return self._nbLignes

    def getNbColonnes(self):
        return self._nbColonnes

    def getVal(self,lig,col):
        return self._listeDesValeurs[lig*self._nbColonnes+col]

    def setVal(self,lig,col,val):
        self._listeDesValeurs[lig*self._nbColonnes+col]=val
```

Avant de détailler cette version, on peut déjà remarquer que ces deux codes se ressemblent fortement, seuls quelques mots *bizarres* comme `class`, `object`, `__init__` et `self` ont été introduits. Nous allons essayer de comprendre ces nouveaux mots clés.

- `class` une classe est la structure qui va englober la structure de données et les opérations sur cette structure. Le nom de la classe qui est définie dans le code est `matrice`. Par la suite chaque instance de cette classe (c'est-à-dire chaque variable qui contiendra concrètement une matrice) s'appellera un *objet*.
- `object` mis entre parenthèses désigne le nom de la classe *object* qui est la classe mère de toutes les autres en Python. On ne va pas entrer dans les détails et considérer que cela fait partie de la syntaxe de la déclaration.
- `self` est un mot clé qui permet de désigner l'objet sur lequel on effectue une opération. Il correspond la plupart du temps au premier paramètre des fonctions de l'API. Dans la suite, les fonctions définies dans la classe qui ont comme premier paramètre `self` s'appelleront des *méthodes*.
- `__init__` est la méthode qui correspond à la fonction `newMatrice`. C'est une méthode particulière car elle permet de créer un nouvel objet de la classe, c'est pour cela que son nom est particulier `__init__`. Dans le vocabulaire de la programmation orientée objet ce type de méthodes s'appelle un *constructeur*. Dans le code de ce constructeur, on va définir la structure de données qui contiendra les données. Ici la structure de triplet a été remplacée par trois "sous-variables" de `self` que l'on appelle *propriétés* ou *attributs*. Il s'agit de `self._nbLignes`, `self._nbColonnes` et `self._listeDesValeurs`. Ces propriétés sont initialisées par le constructeur en utilisant les valeurs des paramètres. Le constructeur ne fait pas de `return` mais on peut considérer que `self` est sa valeur de retour.
- Pour les autres méthodes, on peut simplement remarquer que le paramètre `matrice` de l'API a été remplacé par le paramètre `self` et que pour accéder aux valeurs des attributs de `self` il suffit d'écrire `self.nomAttribut`.

On vient de voir comment on peut faire évoluer une API de programmation vers une classe en programmation orientée objet mais comment cela se passe-t-il au niveau de l'utilisation ? Voici un programme Python qui utilise l'API des matrices et son correspondant en version

programmation orientée objet.

```
from matrice import *
m=newMatrice(5,4,0)
k=0
for i in range(getNbLignes(m)):
  for j in range(getNbColonnes(m)):
    setVal(m,i,j,k)
    k+=1
```

```
from matriceOO import *
m=Matrice(5,4,0)
k=0
for i in range(m.getNbLignes()):
  for j in range(m.getNbColonnes()):
    m.setVal(i,j,k)
    k+=1
```

Les deux principales différences se situent dans la manière dont on crée une matrice et dans la manière d'appeler les méthodes. Dans le premier code, on appelle la fonction `newMatrice` alors que dans le second on appelle le constructeur non pas par son nom mais par le nom de la classe `m=Matrice(5,4,0)`. Pour l'appel des méthodes, le 1er paramètre de chaque méthode (`self` dans la déclaration) est spécifié avec la syntaxe `m.nomMethode(...)`. Par exemple `m.setVal(i,j,k)` signifie qu'on applique la méthode `setVal` de la classe `matrice` sur l'objet `m`. Donc `self` est remplacé par `m` lors de l'appel. La classe est sous-entendue puisque c'est la classe de `m` obligatoirement.

Exercice 1 *Compléter la classe matrice*

Ce premier exercice consiste à transformer les deux fonctions mises en commentaires dans le fichier matriceoo.py en méthodes de la classe matrice et de faire des tests dans le programme principal pour vérifier que vos méthodes marchent bien. Attention les deux fonctions sont écrites avec l'API des matrices, il faut donc modifier leur code pour qu'elles utilisent les méthodes de la classe `matrice`.

Exercice 2 *Gestion des meilleurs scores d'un jeu*

L'objectif de cet exercice est d'implémenter une classe de gestion des meilleurs scores d'un jeu. Pour cela, nous allons implémenter deux classes `score` et `classement`.

A. La classe score

Le constructeur de cette classe prend deux paramètres : le nom du joueur et le nombre de points obtenus. Cette classe ne contient que deux autres méthodes `getNom()` qui retourne le nom du joueur à qui appartient le score et `getPoints()` qui retourne le nombre de points obtenus pour ce score.
Ainsi si on a écrit `s=score('Batman',1236)`, l'appel `s.getNom()` devra retourner `'Batman'` et l'appel `s.getPoints()` devra retourner `1236`.

B. La classe classement

Cette classe gère un classement des meilleurs scores mais en limitant le nombre d'éléments dans ce classement. Lorsque le classement est plein et qu'un nouveau score est ajouté, si ce score est strictement supérieur à l'un des scores du classement le dernier score du classement sera éliminé et le nouveau score sera ajouté à ce classement. Sinon, le score ne sera pas ajouté du tout. Pour les scores qui ont le même nombre de points, c'est l'ordre chronologique qui les départagera (c'est-à-dire c'est le score le plus ancien qui sera avant le plus récent). Voici la liste des méthodes que l'on veut implémenter pour cette classe (à vous de choisir la représentation interne du classement).

1. Le constructeur prend en paramètre le nombre de places disponibles dans le classement. Par exemple `c=classement(5)` indique que l'on ne garde que les 5 meilleurs scores. Le résultat sera un classement vide.

2. `getNbPlaces()` qui retourne le nombre de places maximum dans le classement.
3. `getScore(i)` retourne le score (l'objet) correspondant à la place `i`. S'il n'y a pas de place `i` dans le classement, la méthode retourne `None`.
4. `ajouter(score)` permet d'ajouter un nouveau score au classement en suivant les règles fixées précédemment. La méthode retourne la place du nouveau score dans le classement (-1 si le score n'a pas été ajouté).
5. `affiche()` permet d'afficher le classement.