TP 4 – Classes
Année 2019-2020

Objectifs du TP : Classes et Types Abstraits de Données (TAD)

Exercice 1

Ensemble – Un ensemble est une collection non ordonnée d'objets, contrairement aux séquences comme les listes et les tuples dans lesquels chaque élément est indexé. Un ensemble ne peut pas contenir de doublon : on ne peut y trouver des éléments que une ou zéro fois.

On considère la classe *inSet* qui représente des ensembles d'entiers (définie ci-dessous en Python). Chaque ensemble est donc une instance de la classe *inSet*. Les éléments d'un ensemble sont représentés par une liste d'entiers. Chaque entier contenu dans l'ensemble est représenté une seule fois. On donne ci-dessous la classe *inSet* avec les méthodes suivantes :

- *__init__* : permet de créer un ensemble vide (liste vide)
- *insert* : permet d'insérer un élément e dans l'ensemble
- *member* : permet de vérifier qu'un élément e appartient à l'ensemble
- *remove* : permet de supprimer un élément e contenu dans l'ensemble

```
class inSet(object):
    """An inSet is a set of integers
    The value is represented by a list of ints, self.vals.
    Each int in the set occurs in self.vals exactly once."""

    def __init__(self):
        """Create an empty set of integers"""
        self.vals = []

    def insert(self, e):
        """Assumes e is an integer and inserts e into self"""
        if not e in self.vals:
            self.vals.append(e)

    def member(self, e):
        """Assumes e is an integer
           Returns True if e is in self, and False otherwise"""
        return e in self.vals

    def remove(self, e):
```

```
"""Assumes e is an integer and removes e from self
   Raises ValueError if e is not in self"""
try:
    self.vals.remove(e)
except:
    raise ValueError(str(e) + ' not found')
```

Écrivez les fonctions suivantes en Python :

1. Créez un ensemble E contenant les éléments 1,2,3,4 en utilisant les méthodes définies dans la classe *inSet* .

2. Écrivez la fonction $\_\_str\_\_$ qui permet d'afficher un ensemble de la façon suivante :

   ```
   print(E)
   {1,2,3,4}
   ```

3. Écrivez la fonction *intersect* qui retourne le sous-ensemble intersection entre deux ensembles s_1 et s_2. Par exemple, si s1 = {1,3,4,5} et s2 = {1,4,6,7}, s1.intersect(s2) retournera l'ensemble {1,4}.

4. Écrivez la fonction *union* qui retourne le sous-ensemble union entre deux ensembles s_1 et s_2. Par exemple, si s1 = {1,3,4,5} et s2 = {1,4,6,7}, s1.union(s2) retournera l'ensemble {1,3,4,5,6,7}.

5. Écrivez la fonction *difference* qui prend en argument deux ensembles s_1 et s_2 et retourne les éléments de s_1 non contenus dans s_2 : s1.difference(s2) = s1 - s2. Par exemple, si s1 = {1,3,4,5,8} et s2 = {5,7,8,9}, s1 - s2 retournera {1,3,4}.

6. Écrivez la fonction différence symétrique *diff_sym* qui prend en argument deux ensembles et retourne les éléments contenus dans un seul ensemble à la fois. Par exemple, si s1 = {1,3,4,5,8} et s2 = {5,7,8,9}, s1.diff_sym(s2) retournera {1,3,4,7,9}.
   Écrivez une autre fonction différence symétrique *diff_sym_bis*, soit s1$\Delta s2 = (s1 \cup s2) - (s1 \cap s2)$.

Exercice 2

Nombres complexes –

Dans cet exercice on s'intéresse aux nombres complexes. On considère la classe Complex qui comprend les méthodes suivantes.

- $\_\_init\_\_$: permet de créer un complexe à partir de deux réels représentant respectivement la partie réelle *real* et la partie imaginaire *imag* du complexe ;
- $\_\_add\_\_$: permet d'additionner 2 nombres complexes ;
- $\_\_sub\_\_$: permet de soustraire 2 nombres complexes ;
- $\_\_mul\_\_$: permet de multiplier 2 nombres complexes ;
- $\_\_div\_\_$: permet de diviser 2 nombres complexes ;

- __*module*__ : permet de calculer le module d'un nombre complexe ;
- __*str*__ : permet d'afficher un nombre complexe ;

Écrivez les fonctions suivantes en Python. Vous testerez au fur et à mesure chacune des méthodes de la classe *Complex*.

1. Implémentez la classe *Complex*.
2. Modifiez la fonction __*init*__ pour qu'elle prenne 3 paramètres au lieu de deux, les 2 premiers étant : a) soit les parties réelle *real* et imaginaire *imag* du complexe, soit les module *rho* et argument *theta* du complexe en notation polaire. Le 3ème argument est un booléen dont la valeur sera True si le complexe est en notation cartésienne, soit False si le complexe est en notation polaire.
   La méthode __*init*__ aura ainsi 4 attributs: *real*, *imag*, *rho*, *theta*.
   Par exemple, on peut créer ainsi des instances de la classe Complexe :
   z = Complex(1.,2.,True) crée le complexe $z = 1 + i2$
   w = Complex(1.4,1.5,False) crée le complexe $z = 1.4.exp(i.1.5)$
3. Écrivez la fonction *to_polaire* qui permet de transformer un complexe défini en notation cartésienne en un complexe défini en notation polaire.
   On rappelle que $z = x + iy = \rho.$ exp(i θ)
4. Écrivez la fonction *print_polaire* qui permet d'afficher un nombre complexe défini en notation polaire.
5. Écrivez la fonction *to_cartesien* qui permet de transformer un complexe défini en notation polaire en un complexe défini en notation cartésienne.

Exercice 3

Vecteurs –

Dans cet exercice on s'intéresse aux vecteurs dans l'espace euclidien de dimension 3. On considère la classe Vecteur qui comprend les méthodes suivantes que vous implémenterez. Pour chaque méthode il faut prévoir un jeu de test.

- __*init*__ : permet de créer un vecteur à partir de trois réels x, y, z ;
- __*add*__ : permet d'additionner 2 vecteurs, en retournant un nouveau vecteur ; vous utiliserez ensuite l'opérateur + pour additionner deux vecteurs $v1$ et $v2$: $v1 + v2$;
- __*iadd*__ : permet d'additionner 2 vecteurs, en modifiant le vecteur en place ; vous utiliserez ensuite l'opérateur += pour additionner les vecteurs v et $v1$: $v+ = v1$;
- Écrivez la méthode *mult_scalaire* qui permet de multiplier un vecteur par un scalaire ;
- Écrivez la méthode *norme* qui retourne la norme d'un vecteur v ;
- Écrivez la méthode *normaliser* qui normalise le vecteur v ;
- Écrivez la méthode *produit_scalaire* qui retourne le produit scalaire de l'instance courante avec le Vecteur passé en paramètre ;
- Écrivez la méthode *produit_vectoriel* qui retourne un nouveau vecteur qui est le produit vectoriel de l'instance courante par le vecteur passé en paramètre ;

- Écrivez la méthode *est_ orthogonal* qui retourne une valeur booléenne indiquant si l'instance courante du vecteur est orthogonale au vecteur passé en paramètre ;
- Écrivez la méthode *est_ colineaire* qui retourne un booléen indiquant si l'instance courante du vecteur est colinéaire au vecteur passé en paramètre.