

## N. S. I. Term. - DS04 (lundi 27 novembre 2023)

### Exercice 1.

On considère une classe `Fraction` dont la définition est la suivante :

```
class Fraction:
    def __init__(self, n, d):
        self.num = n
        self.den = d
```

**Q1.** Rappeler par quelle instruction on peut définir un objet `f`, de type `Fraction`, représentant la fraction  $\frac{2}{3}$ .

**Q2.** Modifier la définition de la classe `Fraction`, de sorte à s'assurer, à l'aide d'une instruction `assert`, que le dénominateur d'une fraction que l'on instancie n'est pas nulle.

**Q3.** Définir une méthode `affiche`, telle qu'un appel `f1.affiche()` affiche le numérateur de la fraction, suivi de son dénominateur, séparé par un « / ». Par exemple pour la fraction  $\frac{2}{3}$ , on aura comme affichage « 2 / 3 ».

**Q4.** Définir une méthode `simplifiable_par`, telle qu'un appel `f.simplifiable_par(k)` renvoie le booléen `True` si la fraction `f` est simplifiable par `k` et `False` sinon.

On rappelle qu'une fraction  $\frac{a}{b}$  est simplifiable par `k` si `k` divise `a` et `b`.

**Q5.** Définir une méthode `addition`, telle qu'un appel `f1.addition(f2)` renvoie un objet de type `Fraction` représentant une fraction égale à la somme des fractions `f1` et `f2`.

On rappelle la formule d'addition suivante :

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}.$$

**Q6.** Définir une fonction `pgcd(a, b)` qui, si `a` et `b` sont deux entiers strictement positifs, renvoie le plus grand entier divisant à la fois `a` et `b` parmi les entiers de l'intervalle  $\llbracket 1, \min(a, b) \rrbracket$ .

On ne devra pas utiliser la fonction prédéfinie `min`.

**Q7.** Définir une méthode `simplifier`, tel que l'appel `f.simplifier()` modifie le numérateur et le dénominateur de la fraction `f`, en les divisant par le pgcd de `a` et `b`. On supposera que la fonction `pgcd` a été définie.

### Exercice 2.

Pour chacune des questions 1. et 3. un constructeur initialisant les attributs des objets devra être défini. On suppose qu'une unité de mesure a été choisie, dans laquelle sont exprimées toutes les mesures de longueur.

**1.** Définir en Python une classe nommée `Carre`, dont les objets, représentant des carrés, possèdent chacun un seul attribut, noté `c`, qui est sa mesure de côté, et munie de deux méthodes :

- une méthode `perimetre`, permettant de retourner le périmètre du carré ;
- une méthode `aire`, permettant de retourner son aire.

**2.** Donner une instruction permettant de définir un carré de côté 5, puis une instruction permettant d'afficher son périmètre et son aire.

**3.** Définir en Python une classe nommée `Triangle`, dont les objets, représentant des triangles, ont pour attributs les mesures des trois côtés d'un triangle, et munie de trois méthodes :

- une méthode `perimetre`, permettant de retourner le périmètre du triangle ;

- une méthode `aire`, permettant de retourner son aire, calculée à l'aide de la méthode de Héron, par la formule  $\mathcal{A} = \sqrt{p(p-a)(p-b)(p-c)}$  où  $p = \frac{a+b+c}{2}$ .
- une méthode `est_rectangle`, renvoyant `True` si le triangle est rectangle et `False` sinon.

4. Donner une instruction permettant de définir un triangle de côtés 5,4 ; 9 et 7,2, et puis une instruction permettant d'afficher son périmètre, son aire et s'il est rectangle ou non.

### Exercice 3.

On considère la classe `mystere`, définie de la façon suivante :

```
class mystere:
    def __init__(self):
        self.Liste = []

    def add(self, x):
        if len(self.Liste) == 0 or x >= self.Liste[len(self.Liste)-1]:
            self.Liste.append(x)
        elif x < self.Liste[0]:
            self.Liste = [x] + self.Liste
        else:
            for i in range(len(self.Liste)):
                if x > self.Liste[i] and x <= self.Liste[i + 1]:
                    self.Liste = self.Liste[:i+1] + [x] + self.Liste[i+1:]
                    break
```

On rappelle que l'instruction `break` (dont l'utilisation est déconseillée en Python) permet de terminer prématurément une boucle (comme le fait une instruction `return`, lorsqu'elle est exécutée dans une boucle dans une fonction).

1. Que fait le programme suivant ? (répondre en une ou deux lignes – on détaillera en question 2.)

```
L = mystere()

for i in [78, 89, 10, 50, 7]:
    L.add(i)

print(L.Liste)
```

2. Détailler ensuite :

- le rôle de la première instruction ;
- l'état de `L` à la fin de la première itération de la boucle `for` ;
- l'état de `L` au début et à la fin de la troisième itération de la boucle `for` ;
- de quel algorithme connu se rapproche ce que réalise ici l'instruction `L.add(i)` à chaque itération ?

3. Ajouter à la classe `mystere` une méthode `ecc`, qui crée et renvoie une liste dont chaque élément est la somme de cet élément et de tous ceux qui le précèdent.

Par exemple, si l'attribut `Liste` de l'objet vaut `[1, 2, 4, 5]`, l'appel à la méthode `ecc` doit renvoyer un objet dont l'attribut `Liste` a pour valeur `[1, 3, 7, 12]`.

**Exercice 4.**

Dans cet exercice, on appelle carré d'ordre  $n$  un tableau de  $n$  lignes et  $n$  colonnes dont chaque case contient un entier naturel.

Exemples :

1	1
1	1

c2

Un carré d'ordre 2

2	9	4
7	5	3
6	1	8

c3

Un carré d'ordre 3

4	5	16	9
14	7	2	11
3	10	15	6
13	12	8	1

c4

Un carré d'ordre 4

Un carré est dit magique lorsque les sommes des éléments situés sur chaque ligne, chaque colonne et chaque diagonale sont égales. Ainsi c2 et c3 sont magiques car la somme de chaque ligne, chaque colonne et chaque diagonale est égale à 2 pour c2 et 15 pour c3. c4 n'est pas magique car la somme de la première ligne est égale à 34 alors que celle de la dernière colonne est égale à 27.

La classe Carre ci-après contient des méthodes qui permettent de manipuler des carrés.

Compléter la fonction `est_magique` qui prend en paramètre un carré et qui renvoie la valeur de la somme si ce carré est magique, `False` sinon.

```
class Carre:
    def __init__(self, tableau = []):
        self.ordre = len(tableau)
        self.valeurs = tableau

    def affiche(self):
        '''Affiche un carré'''
        for i in range(self.ordre):
            print(self.valeurs[i])

    def somme_ligne(self, i):
        '''Calcule la somme des valeurs de la ligne i'''
        return sum(self.valeurs[i])

    def somme_col(self, j):
        '''Calcule la somme des valeurs de la colonne j'''
        return sum([self.valeurs[i][j] for i in range(self.ordre)])

def est_magique(carre):
    n = carre.ordre
    s = carre.somme_ligne(0)

    #test de la somme de chaque ligne
    for i in range(..., ...):
        if carre.somme_ligne(i) != s:
            return ...

    #test de la somme de chaque colonne
    for j in range(n):
        if ... != s:
            return False

    #test de la somme de chaque diagonale
    if sum([carre.valeurs[...][...] for k in range(n)]) != s:
        return False
    if sum([carre.valeurs[k][n-1-k] for k in range(n)]) != s:
        return False

    return ...
```