Exercice 1 — Différentes implémentations

Voici un programme écrit en Python :

```
l = [1,4,7,18,3,42]
tmp = 0
for e in l:
    tmp += e
print(tmp/len(l))
```

1 ▶ Quel est le style de programmation utilisé plus haut ?

2 ▶ Réécrire ce programme en style fonctionnel.

3 ▶ Réécrire ce programme en style orienté objet.

4 ▶ Expliquer les différents avantages et inconvénients de ces différents styles.

Exercice 2 — Programmation orienté objet en Python

Voici une classe écrite en Python destinée à travailler avec les nombres rationnels (quotients des nombres relatifs) :

```
def pgcd(a,b):
    while b<>0:
        c = b
        b = a%b
        a = c
    return(a)

class Rational:
    """Rational numbers class."""

    def __init__(self, num, den):
        self.num = num
        self.den = den

    def __str__(self):
        return("{} / {}".
            format(self.num, self.den))

    def approx(self,dp):
        n = self.num
        d = self.den
        s = str(n//d)+"."
        n = n%d
        for i in range(0,dp):
            s+=chr((n*10)//d+48)
            n = (n*10)%d
        return(s)

    def __add__(self, other):
        den = self.den * other.den
        num = (self.num * other.den + ???)
        return(Rational(num, den))

    def __mul__(self, other):
        num = self.num * other.num
        den = ???
        return(Rational(num, den))

    def __div__(self, other):
        num = self.num * other.den
        den = ???
        return(Rational(num, den))

    def simp(self):
        if self.den==0:
            raise(ZeroDivisionError)
        else:
            return(Rationnel(???))
```

1 ▶ Que désigne le mot-clef `self` dans le code ci-dessus ? Comment s'appelle la méthode `__init__` ?

2 ▶ En Python, les méthodes `__init__` et `__str__` doivent absolument être présente dans la définition d'une classe. Expliquer le rôle des autres méthodes en ajoutant des commentaires à ce code.

3 ▶ Compléter les méthodes pour qu'elles fassent ce qu'elle doivent faire.

4 ▶ Tester les sorties des instructions suivantes :

```
import rational_class as rational
print("%.100f"%(2/7))
r = rational.Rational(2,7)
print(r.approx(100))
```

Expliquer les différences en détails..

5 ▶ Même question pour le code suivant :

```
print("%.100f"%(0.1+0.2))
a = rational.Rational(1,10)
b = rational.Rational(2,10)
a = a+b
print(a.approx(100))
```