Classes

Programmation orientée objet

Prérequis : cours et exercices sur la programmation orientée objet.

Exercice 1 *Intervalles*

Le but de cette partie est de construire la classe `Intervalle`, représentant des intervalles de nombres réels. La classe possède deux attributs `borne_inf` et `borne_sup` qui représentent respectivement la borne inférieure et la borne supérieure de l'intervalle. Ces deux bornes seront toujours considérées comme incluses.

Si `borne_sup < borne_inf`, l'intervalle est considéré comme vide.

1. Programmer le constructeur de la classe. Il aura deux paramètres `a` et `b` correspondant respectivement à `borne_inf` et `borne_sup`.
2. Écrire une méthode `est_vide` qui renvoie `True` si l'intervalle est vide, `False` sinon.
3. Écrire une méthode `affichage` qui renvoie une représentation de l'intervalle sous forme de chaîne de caractères : `"[borne_inf; borne_sup]"`. Si l'intervalle est vide, l'affichage sera : `"[ ]"`.
4. Écrire une méthode `longueur` qui renvoie l'amplitude de l'intervalle. Si l'intervalle est vide, la méthode renvoie `None`.
5. Écrire une méthode `contient` de paramètre `x` qui renvoie `True` si le réel `x` appartient à l'intervalle, `False` sinon.
6. Écrire une méthode `equivaut`, de paramètre une (autre) instance de la classe `Intervalle`, qui renvoie `True` s'ils sont égaux, `False` sinon. Tous les intervalles vides sont considérés comme égaux.
7. Écrire une méthode `inclus`, de paramètre une instance de `Intervalle`, qui renvoie `True` si l'intervalle est inclus dans l'autre, `False` sinon.
8. **Bonus 1.** Écrire une méthode `union`, de paramètre une instance de la classe `Intervalle`, qui renvoie une instance de la classe `Intervalle` dont les bornes correspondent au plus petit intervalle contenant la réunion des deux intervalles.
9. **Bonus 2.** Écrire une méthode `intersection`, de paramètre une instance de la classe `Intervalle`, qui renvoie une instance de la classe `Intervalle` dont les bornes correspondent à l'intersection entre les deux intervalles.

Corrigé de l'exercice 1.

```
class Intervalle:
    def __init__(self, a, b):
        self.borne_inf = a
        self.borne_sup = b

    def est_vide(self):
        return self.borne_sup < self.borne_inf

    def affichage(self):
        if self.est_vide():
```

```
            return "[ ]"
        return f"[{self.borne_inf}; {self.borne_sup}]"

    def longueur(self):
        if self.est_vide():
            return None
        return self.borne_sup - self.borne_inf

    def contient(self, x):
        return self.borne_inf <= x <= self.borne_sup

    def equivaut(self, autre):
        if self.est_vide() and autre.est_vide():
            return True
        return self.borne_inf == autre.borne_inf and self.borne_sup == autre.borne_sup

    def inclus(self, autre):
        if self.est_vide():
            return True
        return autre.borne_inf <= self.borne_inf and self.borne_sup <= autre.borne_sup

    def union(self, autre):
        if self.est_vide():
            return Intervalle(autre.borne_inf, autre.borne_sup)
        if autre.est_vide():
            return Intervalle(self.borne_inf, self.borne_sup)
        inf = min(self.borne_inf, autre.borne_inf)
        sup = max(self.borne_sup, autre.borne_sup)
        return Intervalle(inf, sup)

    def intersection(self, autre):
        inf = max(self.borne_inf, autre.borne_inf)
        sup = min(self.borne_sup, autre.borne_sup)
        return Intervalle(inf, sup)

i = Intervalle(1, 4)
j = Intervalle(2, 6)
print(i.affichage())                    # [1; 4]
print(Intervalle(5, 2).affichage())     # [ ]
print(i.longueur())                     # 3
print(i.contient(4))                    # True
print(i.inclus(j))                      # False
print(i.union(j).affichage())           # [1; 6]
print(i.intersection(j).affichage())    # [2; 4]
```

Les méthodes `est_vide`, `contient`, `equivaut` et `inclus` renvoient directement le résultat d'une comparaison, qui vaut `True` ou `False`. Le cas de l'intervalle vide se traite avant le cas général. `union` et `intersection` construisent un nouvel objet `Intervalle` : les intervalles de départ ne sont pas modifiés. Une intersection vide donne un intervalle dont la borne supérieure est plus petite que la borne inférieure, donc un intervalle vide.

Exercice 2 *Fractions*

Le but de cette partie est de construire la classe `Fraction`, représentant le quotient de deux nombres relatifs. La classe possèdera deux attributs `numerateur` et `denominateur`.

1. Programmer le constructeur de la classe. Il aura deux paramètres `a` et `b` correspondant respectivement à `numerateur` et `denominateur`. On vérifiera les préconditions par des assertions : par exemple, `assert b != 0, "le dénominateur ne doit pas être nul"`.
2. Écrire une méthode `affichage` qui renvoie une représentation de la fraction sous forme de chaîne de caractères : `"numerateur/denominateur"`. Si le dénominateur vaut 1 ou -1, l'affichage ne comportera pas de `"/"`.
3. Écrire une méthode `simplifie` qui modifie les attributs `numerateur` et `denominateur` pour que la fraction soit irréductible. Si la fraction est négative, c'est le `numerateur` qui est négatif. Le pgcd se calcule dans cette méthode.
4. Écrire une méthode `egal` de paramètre une instance de la classe `Fraction`, qui renvoie `True` si les deux fractions sont égales, `False` sinon.
5. Écrire une méthode `produit` de paramètre une instance de la classe `Fraction`, qui renvoie une instance de la classe `Fraction` correspondant au produit des deux fractions. La nouvelle instance sera irréductible.
6. Écrire une méthode `quotient`, de paramètre une instance de la classe `Fraction`, qui renvoie une instance de la classe `Fraction` correspondant au quotient des deux fractions. La nouvelle instance sera irréductible.
7. Écrire une méthode `somme` de paramètre une instance de la classe `Fraction`, qui renvoie une instance de la classe `Fraction` correspondant à la somme des deux fractions. La nouvelle instance sera irréductible.

Corrigé de l'exercice 2.

```
class Fraction:
    def __init__(self, a, b):
        assert b != 0, "le dénominateur ne doit pas être nul"
        self.numerateur = a
        self.denominateur = b

    def affichage(self):
        if self.denominateur == 1:
            return str(self.numerateur)
        if self.denominateur == -1:
            return str(-self.numerateur)
        return f"{self.numerateur}/{self.denominateur}"

    def simplifie(self):
        a, b = abs(self.numerateur), abs(self.denominateur)
        while b != 0:
            a, b = b, a % b
        self.numerateur = self.numerateur // a
        self.denominateur = self.denominateur // a
        if self.denominateur < 0:
            self.numerateur = -self.numerateur
            self.denominateur = -self.denominateur

    def egal(self, autre):
        a, b = self.numerateur, self.denominateur
        c, d = autre.numerateur, autre.denominateur
        return a * d == b * c

    def produit(self, autre):
        a, b = self.numerateur, self.denominateur
        c, d = autre.numerateur, autre.denominateur
```

```
        f = Fraction(a * c, b * d)
        f.simplifie()
        return f

    def quotient(self, autre):
        a, b = self.numerateur, self.denominateur
        c, d = autre.numerateur, autre.denominateur
        f = Fraction(a * d, b * c)
        f.simplifie()
        return f

    def somme(self, autre):
        a, b = self.numerateur, self.denominateur
        c, d = autre.numerateur, autre.denominateur
        f = Fraction(a * d + b * c, b * d)
        f.simplifie()
        return f

f = Fraction(6, -8)
f.simplifie()
print(f.affichage())                                    # -3/4
print(Fraction(1, 2).egal(Fraction(2, 4)))              # True
print(Fraction(2, 3).produit(Fraction(3, 4)).affichage())  # 1/2
print(Fraction(2, 3).quotient(Fraction(4, 9)).affichage()) # 3/2
print(Fraction(1, 6).somme(Fraction(1, 3)).affichage())    # 1/2
```

`simplifie` modifie l'objet et ne renvoie rien ; elle divise le numérateur et le dénominateur par leur pgcd, calculé par l'algorithme d'Euclide sur les valeurs absolues, puis reporte le signe sur le numérateur. `egal` compare les produits en croix, sans simplifier. `produit`, `quotient` et `somme` renvoient une nouvelle fraction, simplifiée avant d'être renvoyée. Pour `quotient`, si la seconde fraction est nulle, le constructeur refuse la nouvelle fraction, dont le dénominateur serait nul.