Python – Programmation orientée objet

Les fractions

Pour pouvoir faire les exercices suivants, il faut copier dans vos documents le fichier `fractions_objet.py` qui se trouve dans `Echange`.
Pour créer une fraction avec cette classe, il faut faire :

```
>>> Fraction(12, 18)
Fraction(2, 3)
>>> f = Fraction(12, 18)
>>> f
Fraction(2, 3)
```

On peut remarquer que les fractions sont bien simplifiées.
Afin de connaître les méthodes disponibles dans ce module, on peut utiliser la commande suivante :

```
>>> help(Fraction)
Help on class Fraction in module __main__:

class Fraction(builtins.object)
 |  Fraction(n, d)
 |
 |  Classe permettant de représenter des fractions.
 |  Permet de faire des calculs exactes.
 |
 |  Methods defined here:
 |
 |  __add__(self, other)
 |      Renvoie une fraction irreductible égale à
 |      f1 + other et dont le dénominateur est positif.
...
```

On peut voir dans la liste les méthodes `numerateur(self)` et `denominateur(self)`, mais pas `_simplifie(self)` puisqu'elle commence par `_`. Elle n'apparaît pas dans l'interface, mais elle est quand même accessible si on connaît son nom. On remarque aussi que les attributs n'apparaîssent pas non plus.
Dans la mesure du possible, on préférera utiliser les méthodes `numerateur(self)` et `denominateur(self)` plutôt que les attributs `n` et `d`.

Exercice 1 : Rajouter à la fin du fichier deux fractions qui pourront servir pour les tests : $f1 = \frac{4}{3}$ et $f2 = \frac{-7}{2}$.

L'affichage des fractions est obtenue si on tape une expression qui renvoie une fraction. Cet affichage est fait par la méthode `__repr__` qui a pour but de renvoyer une expression permettant de recréer l'objet. Si cette méthode n'est pas définie dans la classe, on obtient l'adresse de l'objet.

```
>>> Fraction(3, 4)
<__main__.Fraction object at 0x7f4b72186580>
```

Par contre, on ne peut pas utiliser **print** pour afficher une fraction. Pour cela, il faut définir la méthode **__str__**.

Exercice 2 : Compléter la méthode **__str__(self)** qui renvoie un texte de la forme 'n / d' ou 'n' si le dénominateur vaut 1. Il est alors possible d'écrire **str**(objet) et **print**(objet).

```
>>> print(Fraction(25, 5))
5
>>> str(f1)
'4 / 3'
```

Exercice 3 : Compléter les méthodes suivantes :

Méthode	Description
decimal(self)	Renvoie la valeur décimale, approchée si nécessaire, de la fraction.
est_entier(self)	Renvoie un booléen indiquant si la fraction est égale à un nombre entier ou pas.
inverse(self)	Renvoie une fraction irreductible égale à l'inverse de la fraction et dont le dénominateur est positif.

```
>>> f2.decimal()
-3.5
>>> f2.est_entier()
False
>>> Fraction(25, 5).est_entier()
True
>>> f2.inverse()
Fraction(-2, 7)
```

Exercice 4 : Compléter les méthodes suivantes :

Méthode	Opérateur	Description
__neg__(self)	-f	Renvoie une nouvelle fraction égale à l'opposé de la fraction.
__add__(self, other)	+	Renvoie une nouvelle fraction égale à la somme des deux autres.
__sub__(self, other)	-	Renvoie une nouvelle fraction égale à la différence des deux autres.
__mul__(self, other)	*	Renvoie une nouvelle fraction égale au produit des deux autres.
__truediv__(self, other)	/	Renvoie une nouvelle fraction égale à la division des deux autres.
__pow__(self, k)	*	Renvoie une nouvelle fraction égale à la fraction à la puissance k.

Quelques rappels de mathématiques :

$$\frac{a}{b}+\frac{c}{d}=\frac{ad+bc}{bd} \qquad \frac{a}{b}-\frac{c}{d}=\frac{ad-bc}{bd} \qquad \frac{a}{b}\times\frac{c}{d}=\frac{ac}{bd} \qquad \frac{a}{b}:\frac{c}{d}=\frac{a}{b}\times\frac{d}{c} \qquad \left(\frac{a}{b}\right)^k=\frac{a^k}{b^k}$$

Ces méthodes permettent d'écrire, par exemple f1+f2 à la place de f1.__add__(f2).
De même, on peut écrire -f1, f1-f2, f1*f2, f1/f2 ou f1**k.

```
>>> -f2
Fraction(7, 2)
>>> -f1
Fraction(-4, 3)
>>> f1 + f2
Fraction(-13, 6)
>>> f1 - f2
Fraction(29, 6)
>>> f1 * f2
Fraction(-14, 3)
>>> f1 / f2
Fraction(-8, 21)
>>> f1 ** 3
Fraction(-64, 27)
```

Exercice 5 : Compléter les méthodes suivantes :

Méthode	Opérateur	Description
`__eq__(self, other)`	==	Renvoie un booléen correspondant à l'égalité entre les deux fractions.
`__lt__(self, other)`	<	Renvoie un booléen correpsondant à `self` < `other`.
`__le__(self, other)`	<=	Renvoie un booléen correpsondant à `self` ≤ `other`.

Pour rappel :

$$\frac{a}{b} = \frac{c}{d} \iff ad = bc \qquad \frac{a}{b} < \frac{c}{d} \iff ad < bc \quad \text{(avec } b > 0 \text{ et } d > 0\text{)}$$

Pour `__le__`, on pourra utiliser les deux autres méthodes.

```
>>> f1 < f2
False
>>> f1 == f2
False
>>> Fraction(5, 6) == Fraction(10, 12)
True
>>> f2 <= f1
True
```

Une fois ces fonctions définies, il est également possible d'utiliser les opérateurs définis de façon implicite, comme >, >= et !=.

```
>>> f1 > f2
True
>>> f1 != f2
True
```

Les heures

Pour les exercices suivants, il faut copier le fichier `heures_objet.py` dans vos documents. La classe `Heure` permet de représenter une heure de la journée. Les objets possèdent 3 attributs : `h`, `m` et `s` qui correspondent aux heures, minutes et secondes.

Ainsi, l'heure 14:26:17 sera représentée par `Heure(14, 26, 17)`. Les heures doivent donc aller de 00:00:00 à 23:59:59.

Exercice 6 : Compléter le constructeur de la classe `Heure`. Il prend comme paramètres, en plus de `self`, 3 entiers positifs `h`, `m` et `s`. Au final, il faut avoir $0 \leq$ `self.h` < 24, $0 \leq$ `self.m` < 60 et $0 \leq$ `self.s` < 60. Si les paramètres ne vérifient pas ces conditions, il faut adapter les valeurs. Par exemple 12:00:70 correspond à 12:01:10, ou encore 23:62:00 deviendra 00:02:00.

Exercice 7 : Compléter la méthode `__repr__(self)` qui renvoie le texte permettant de recréer l'objet `Heure(h, m, s)`.

```
>>> Heure(12, 7, 15)
Heure(12, 7, 15)
>>> Heure(23, 59, 80)
Heure(0, 0, 20)
```

Exercice 8 : Compléter la méthode `__str__(self)` qui renvoie un texte correspondant à l'heure sous le format `'Xh Ym Zs'`, où `X`, `Y` et `Z` sont les valeurs des différents attributs de `self`.

```
>>> print(Heure(12, 7, 15))
12h 7m 15s
>>> print(Heure(23, 59, 80))
0h 0m 20s
```

Exercice 9 : Compléter la méthode `avance(self, s)` qui avance l'heure de l'objet de `s` secondes. Là encore, il faudra s'assurer que les valeurs stockées sont conformes.

```
>>> h = Heure(12, 7, 15)
>>> h.avance(80)
>>> print(h)
12h 8m 35s
```

Les heures 2, le retour

Toujours dans le même fichier, nous allons compléter la classe `Heure2` qui ne stocke que les secondes écoulées depuis 00:00:00.

Exercice 10 : Compléter le constructeur `__init__(self, h, m, s)`, qui convertit l'heure donnée en secondes, qui peuvent donc dépasser 60, mais pas 86 400, le nombre de secondes dans une journée.

```
>>> h = Heure2(20, 17, 35)
>>> h.s
73055
>>> 20*3600 + 17*60 + 35
73055
```

Exercice 11 : Compléter la méthode `_conversion(self)` qui renvoie un triplet `(h, m, s)` qui correspond à l'heure stockée.

```
>>> Heure2(20, 17, 35)._conversion()
(20, 17, 35)
>>> Heure2(20, 70, 85)._conversion()
(21, 11, 25)
```

Exercice 12 : Compléter la méthode `__repr__(self)` qui renvoie le texte permettant de recréer l'objet `Heure2(h, m, s)`.

```
>>> Heure2(20, 17, 35)
Heure2(20, 17, 35)
>>> Heure2(20, 70, 85)
Heure2(21, 11, 25)
```

Exercice 13 : Compléter la méthode `__str__(self)` qui renvoie un texte correspondant à l'heure sous le format `'Xh Ym Zs'`, où `X`, `Y` et `Z`.

```
>>> print(Heure2(20, 17, 35))
20h 17m 35s
```

Exercice 14 : Compléter la méthode `avance(self, s)` qui avance l'heure de l'objet de `s` secondes.

```
>>> h = Heure2(7, 59, 14)
>>> h.avance(420)
>>> h
Heure2(8, 6, 14)
```

Exercice 15 : Compléter la méthode `__eq__(self, other)` qui renvoie un booléen équivalent à `self` = `other`.

```
>>> Heure2(0, 0, 80) == Heure2(0, 1, 20)
True
>>> Heure2(15, 12, 10) == Heure2(15, 12, 11)
False
```

Exercice 16 : Compléter la méthode `copy(self)` qui renvoie un nouvel objet de classe `Heure2` égal à `self`.

```
>>> h = Heure2(17, 0, 0)
>>> h2 = h.copy()
>>> h2
Heure2(17, 0, 0)
>>> h.avance(90)
>>> h
Heure2(17, 1, 30)
>>> h2
Heure2(17, 0, 0)
```

Les dates

Copier le fichier `dates_objet.py` depuis `Echange` vers vos documents.
Ce fichier contient la classe `Date` qui définit une date, qui contient trois attributs : `jour`, `mois` et `annee`.
La classe contient deux **attributs de classe**, `nom_mois` et `b_jours`, qui associent au numéro du mois le nom et le nombre de jours qu'il contient. Contrairement aux attributs des objets, qui sont indépendants entre chaque objet, ceux de la classe sont partagés. Ainsi, si un objet modifie cette valeur, elle l'est pour tous les autres. Il vaut mieux n'utiliser ces attributs

que pour des constantes. Il est possible de le utiliser en dehors de la classe en écrivant `Date.nom_mois[mois]` ou `Date.nb_jours[mois]`. Sinon, à l'intérieur d'un objet, on peut utilier `self.nom_mois[mois]` et `self.nb_jours[mois]`.

Exercice 17 : Compléter le constructeur de la classe `Date`. On suppose que les paramètres donnés sont valides.

```
>>> Date(6, 10, 2021)
Date(6, 10, 2021)
```

Exercice 18 : Compléter la méthode `__str__(self)` qui renvoie un texte correspondnat à la date en indiquant le nom des mois. Par exemple : `'1 septembre 2021'`. Pour cela, vous pouvez utiliser l'attribut de classe `nom_mois`.

```
>>> print(Date(6, 10, 2021))
6 octobre 2021
```

Exercice 19 : Compléter la méthode `__lt__(self, other)` qui permet de comparer deux dates en faisant `d1 < d2`.

```
>>> Date(7, 9, 2020) < Date(8, 9, 2020)
True
>>> Date(7, 9, 2020) < Date(1, 1, 2022)
True
>>> Date(7, 9, 2020) < Date(8, 10, 2019)
False
```

Exercice 20 : Compléter la fonction `_bissextile(annee)` qui renvoie un booléen indiquant si l'année est bissextile ou non. Elle l'est si une des deux conditions suivantes est vérifiée :

- l'année est divisible par 4 mais pas par 100
- l'année est divisible par 400

```
>>> _bissextile(2020)
True
>>> _bissextile(2100)
False
>>> _bissextile(2000)
True
>>> _bissextile(2021)
False
```

Cette fonction est utilisée dans la méthode `bissextile(self)` qui renvoie un booléen indiquant si l'année de la date est bissextile ou pas.

Pour aller plus loin

Exercice 21 : Compléter la méthode `ajoute_un_jour(self)` qui modifie la date en rajoutant un jour.

```
>>> d = Date(6, 10, 2021)
>>> d.ajouter_un_jour()
>>> print(d)
7 octobre 2021
>>> d = Date(31, 12, 2021)
>>> d.ajouter_un_jour()
>>> print(d)
1 janvier 2022
>>> d = Date(28, 2, 2020)
>>> d.ajouter_un_jour()
>>> print(d)
29 février 2020
```

Exercice 22 : Compléter la méthode `ajoute_jours(self, jours)` qui modifie la date en rajoutant `jours` jours.

```
>>> d = Date(6, 10, 2021)
>>> d.ajouter_jours(100)
>>> d
Date(14, 1, 2022)
```

Exercice 23 : Compléter la méthode `nb_jours_depuis_debut_annee(self)` qui renvoie le nombre de jours depuis le début de l'année, en comptant le jour actuel.

```
>>> Date(1, 1, 2021).nb_jours_depuis_debut_annee()
1
>>> Date(6, 10, 2021).nb_jours_depuis_debut_annee()
279
>>> Date(6, 10, 2020).nb_jours_depuis_debut_annee()
280
```

Exercice 24 : Compléter la méthode `difference(self, other)` qui renvoie le nombre de jours qui séparent les deux dates. L'ordre dans lequel sont données les dates n'a pas d'importance. Il faut faire attention aux années bissextiles.

```
>>> d1 = Date(1, 7, 2022)
>>> d2 = Date(6, 10, 2021)
>>> d1.difference(d2)
268
>>> d2.difference(d1)
268
```

Exercice 25 : Déterminer le nombre de jours écoulés depuis votre naissance.