Python – Abstraction et interface

Consignes

L'objectif de cette feuille est de vous faire réaliser l'implémentation d'une interface donnée. Vous devez compléter chacune des fonctions de l'interface, même si vous ne traitez pas tous les cas. Vous êtes libres d'en rajouter d'autres. Vous trouverez le squelette de chacune des fonctions dans le fichier `fractions_prenom_nom.py`, que vous renommerez.
Pour chacune des fonctions de l'interface, il y a le texte de la description en commentaire, après la ligne de définition de la fonction :

```
def fraction(n, d):
    """ Renvoie une fraction irreductible correspondant
    à n/d et dont le dénominateur est positif."""
    ...
    return ...
```

Il est ainsi possible d'obtenir de l'aide sur la fonction :

```
>>> help(fraction)
Help on function fraction in module __main__:

fraction(n, d)
    Renvoie une fraction irreductible correspondant
    à n/d et dont le dénominateur est positif.
```

Pour les fonctions que vous rajouterez, essayez de respecter la convention consistant à mettre un "_" avant leur nom, commme `_ma_fonction` par exemple, si elles servent de fonction intermédiaire. Si au contraire elles peuvent être rajoutées à l'interface, vous les nommerez normalement.
Voici quelques exemples d'utilisation :

```
>>> f1 = fraction(9, 2)
>>> f2 = fraction(-5, 45)
>>> numerateur(f1)
9
>>> denominateur(f1)
2
>>> texte(f2)
'-1/9'
>>> decimal(f1)
4.5
>>> decimal(f2)
-0.1111111111111111
>>> texte(somme(f1, f2))
'79/18'
>>> texte(produit(f1, f2))
'-1/2'
>>> texte(inverse(produit(f1, f2)))
'-2'
>>> egalite(produit(f2, inverse(f2)), fraction(1, 1))
True
```

Fractions

Voici l'interface qui doit correspondre à votre module.

Fonction	Description
`fraction(n, d)`	Renvoie une fraction irreductible correspondant à `n/d` et dont le dénominateur est positif et non nul.
`numerateur(f)`	Renvoie le numérateur de la fraction `f`.
`denominateur(f)`	Renvoie le dénominateur de la fraction `f`.
`decimal(f)`	Renvoie la valeur décimale, approchée si nécessaire, de la fraction `f`.
`est_entier(f)`	Renvoie un booléen indiquant si `f` est égale à un nombre entier ou pas.
`texte(f)`	Renvoie un texte de la forme `'n/d'` ou `'n'` si le dénominateur vaut 1.
`inverse(f)`	Renvoie une fraction irreductible égale à l'inverse de `f` et dont le dénominateur est positif.
`oppose(f)`	Renvoie une fraction irreductible égale à $-$`f` et dont le dénominateur est positif.
`somme(f1, f2)`	Renvoie une fraction irreductible égale à `f1` + `f2` et dont le dénominateur est positif.
`produit(f1, f2)`	Renvoie une fraction irreductible égale à `f1` $\times$ `f2` et dont le dénominateur est positif.
`quotient(f1, f2)`	Renvoie une fraction irreductible égale à `f1` : `f2` et dont le dénominateur est positif.
`egalite(f1, f2)`	Renvoie un booléen équivalent à `f1` = `f2`.
`inferieur(f1, f2)`	Renvoie un booléen équivalent à `f1` < `f2`.

Quelques indications :

- Vous pouvez prendre la représentation interne de votre choix pour les fractions.
- Seules `fraction`, `numerateur` et `denominateur` peuvent utiliser les particularités de votre représentation. Les autres fonctions devront uniqument utiliser les fonctions `fraction`, `numerateur` et `denominateur`.
- Dans un premier temps, vous pouvez ne pas simplifier les fractions.
- La fonction `_pgcd` renvoie le PGCD des deux entiers. C'est le plus grand diviseur commun aux deux entiers. Pour rendre une fraction irréductible, il suffit de diviser le numérateur et le dénominateur par leur PGCD.
- Pour la somme, vous pouvez utiliser la propriété suivante :

$$\frac{n_1}{d_1} + \frac{n_2}{d_2} = \frac{n_1 \times d_2 + n_2 \times d_1}{d_1 \times d_2}$$

- Pour l'égalité, vous ne pouvez pas juste faire `f1 == f2`, mais vous pouvez utiliser l'égalité des numérateurs et dénominateurs, si les fractions sont réduites ou l'égalité des produits en croix :

$$\frac{n_1}{d_1} = \frac{n_2}{d_2} \quad \Leftrightarrow \quad n_1 \times d_2 = n_2 \times d_1$$

- Pour la comparaison, vous pouvez utiliser la propriété suivante, si les dénominateurs sont positifs :

$$\frac{n_1}{d_1} < \frac{n_2}{d_2} \quad \Leftrightarrow \quad n_1 \times d_2 < n_2 \times d_1$$

Test du module

Vous devez également écrire un fichier `tests_interface_fractions_prenom_nom.py` dans lequel vous importerez votre autre fichier afin de le tester.
La première ligne de `tests_interface_fractions_prenom_nom.py` doit être :

```
import fractions_prenom_nom as frac
```

Pour utiliser les fonctions de votre module, il faudra donc appeler `frac.fraction(4, 3)`, `frac.numerateur(f)`... Les tests que vous allez produire pourront être mis en commun pour produire une gande base de tests. Vous pouvez rajouter la fonction suivante :

```
def test_valeur(f, n, d):
    assert frac.numerateur(f) == n and frac.denominateur(f) == d
```

Vous pouvez ensuite faire des fonctions différentes pour tester les différentes fonctions de l'interface pour les tester au fur et à mesure :

```
def tests_num_denom():
    test_valeur(frac.fraction(1, 3), 1, 3)
    test_valeur(frac.fraction(9, 2), 9, 2)
    test_valeur(frac.fraction(13, 421), 13, 421)
```

Vous pouvez faire des fonctions qui testent un grand nombre de fractions :

```
def tests_denom_positif():
    for i in range(-200, 200):
        for j in range(-200, 200):
            if j != 0:
                assert frac.denominateur(frac.fraction(i, j)) > 0
```

Ou encore des fonctions qui testent des fractions au hasard :

```
import random

def tests_denom_positif():
    for i in range(1000):
        n = random.randint(-10000, 10000)
        d = random.randint(-10000, 10000)
        if d != 0:
            assert frac.denominateur(frac.fraction(n, d)) > 0
```

Nombres binaires

Vous allez réaliser l'implémentation de l'interface suivante, permettant de manipuler des nombres entiers représentés en binaire. Vous représenterez les nombres par des chaînes de caractères allant du bit de poids faible au bit de poids fort. Ainsi, $4_{10} = 100_2$ sera représenté par `'001'`. Ainsi, le bit en position i correspond à 2^i. Le nombre 0 pourra être représenté par la chaîne vide.

Voici quelques exemples d'utilisations :

```
>>> b1 = dec_en_bin(147)
>>> b2 = dec_en_bin(35)
>>> longueur(b1)
8
>>> n_ieme_bit(b1, 4)
'1'
>>> texte(somme(b1, b2))
'11111010'
>>> reduire(et(b1, b2))
'1'
>>> bin_en_dec(ou(b1, b2))
249
```

L'interface est la suivante :

Fonction	Description
`dec_en_bin(n)`	Renvoie le nombre binaire correspondant à l'entier positif `n`.
`longueur(b)`	Renvoie le nombre de bits de `b`. Il peut y avoir des 0 inutiles.
`n_ieme_bit(b, n)`	Renvoie le bit de `b`, sous forme de texte, correspondant à 2^n. Si `n` est supérieur à `longueur(b)`, renvoie un `'0'`.
`reduire(b)`	Renvoie l'écriture binaire de `b` où tous les 0 inutiles ont été enlevés.
`bin_en_dec(b)`	Renvoie l'entier décimal égal à `b`.
`texte(b)`	Renvoie la chaîne de caractères correspondant à `b` dans le bon sens.
`somme(b1, b2)`	Renvoie la somme de `b1` et de `b2`.
`produit(b1, b2)`	Renvoie le produit de `b1` et de `b2`.
`et(b1, b2)`	Renvoie le "et" bit à bit entre les deux nombres binaires `b1` et `b2`.
`ou(b1, b2)`	Renvoie le "ou" bit à bit entre les deux nombres binaires `b1` et `b2`.
`non(b)`	Renvoie l'inverse bit à bit de `b`.

Quelques indications :

- Vous ne pouvez utiliser `b[...]` que dans `n_ieme_bit`, et `len(b)` uniquement dans `longueur`. Vous devrez utiliser ces nouvelles fonctions dans les autres.
- Pour `texte(b)`, il faut bien penser à retourner l'ordre des bits. Il serait donc une bonne idée d'utiliser une fonction `_renverse(b, p=0)` qui fait justement cela. Il faut aussi penser au cas de la chaîne vide qui correspond à 0.

Tests du module

Sur le modèle de celui fait pour les fractions, réaliser un fichier permettant de tester votre interface.
Après avoir testé vos fonctions de conversion de décimal en binaire et de binaire en décimal, vous pouvez tester les opérations mathématiques en comparant les résultats obtenus avec les résultats attendus en faisant les calculs sur les entiers.