

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

#
# fichier: fraction.py
# version: 0.5.0
# auteur: Pascal CHAUVIN
# date: 2014/10/28
#
# (tous les symboles non internationaux sont volontairement omis)
#

import string

import sys
sys.path.append('../entier_mod')
sys.path.append('../monome_mod')
sys.path.append('../monome_mod/joli_mod')
sys.path.append('../polynome_mod')
sys.path.append('../rationnel_mod')

import entier as ent
import joli
import monome as mo
import polynome as po
import rationnel as ra

class fraction(object):

    def __init__(self, num =po.polynome_nul(), denom =po.polynome_un(), valide =True):
        """ constructeur """
        valide = valide and (num.est_valide() and denom.est_valide())
        if not valide: # normaliser les param.
            num = po.polynome_err()
            denom = po.polynome_un()

        if valide:
            if denom.est_polynome_nul():
                valide = False
                num = po.polynome_err()
                denom = po.polynome_un()

        self.__num = num
        self.__denom = denom
        self.__valide = valide

    def __repr__(self):
        """ _ """
        return "[fraction:\n__num={0},\n__denom={1},\n__valide={2}\n]\n".\
            format(self.__num, self.__denom, self.__valide)

    def __str__(self):
        """ _ """
        if self.__denom.est_polynome_unite():
            return "{0}".format(self.__num)
        else:
            return "({0})/({1})".format(self.__num, self.__denom)

    def joli(self):
        """ _ """
        # return "({0})/({1})".format(self.__num, self.__denom)

```

```

    self.reduire()

    p, q = self.__num, self.__denom

    u = p.joli()
    v = q.joli()

    if p.nombre_monomes() > 1:
        if (not q.est_polynome_unite()):
            u = "(" + u + ")"

    if q.nombre_monomes() > 1:
        v = "(" + v + ")"

    if q.nombre_monomes() == 1:
        if q.degre() == 1:
            if (not q.valuation().est_un()):
                v = "(" + v + ")"

    if q.est_polynome_unite():
        return "{0}".format(u)
    else:
        return "{0}/{1}".format(u, v)

def est_valide(self):
    """ accesseur """
    return self.__valide

def fixer_valide(self, v):
    """ accesseur """
    self.__valide = v

def lire_num(self):
    """ accesseur """
    return self.__num

def lire_denom(self):
    """ accesseur """
    return self.__denom

def __add__(self, autre):
    """ somme """
    if isinstance(autre, fraction):
        if self.__valide and autre.__valide:
            a = self.__num
            b = self.__denom

            p = autre.__num
            q = autre.__denom

            return fraction(a*q + b*p, b*q)

    return fraction(po.polynome_err(), po.polynome_un())

def __neg__(self):
    """ polynome oppose (inverse pour l'addition) """

```

```

    if self.__valide:
        a = self.__num
        b = self.__denom

        return fraction(-a, b)

    return fraction(po.polynome_err(), po.polynome_un())

def oppose(self):
    """ fraction opposee """
    return self.__neg__()

def __sub__(self, autre):
    """ difference """
    return (self + (-autre))

def __mul__(self, autre):
    """ produit """
    if isinstance(autre, fraction):
        if (self.__valide) and (autre.__valide):
            a = self.__num
            b = self.__denom

            p = autre.__num
            q = autre.__denom

            return fraction(a*p, b*q)

    return fraction(po.polynome_err(), po.polynome_un())

def __pow__(self, autre):
    """ exponentiation """
    if isinstance(autre, int):
        autre = fraction(po.polynome(mo.monome(ra.rationnel(autre))), \
            po.polynome_un())

    if isinstance(autre, fraction):
        if (self.__valide) and (autre.__valide):
            a = self.__num
            b = self.__denom

            p = autre.__num
            q = autre.__denom

            pv = p.valuation().lire_num().lire_valeur()
            qv = q.valuation()

            if (p.degre() == 0) and (q.degre() == 0):

                if pv < 0:
                    a, b = b, a
                    p = -p

                if not qv.est_un():
                    return fraction(po.polynome_err(), po.polynome_un())

                return fraction(a**p, b**p)

    return fraction(po.polynome_err(), po.polynome_un())

```

```

def __truediv__(self, autre):
    """ quotient """
    if isinstance(autre, fraction):
        if (self.__valide) and (autre.__valide):
            a = self.__num
            b = self.__denom

            p = autre.__num
            q = autre.__denom

            if p.est_polynome_nul():
                return fraction(po.polynome_err(), po.polynome_un())

            return fraction(a*q, b*p)

    return fraction(po.polynome_err(), po.polynome_un())

def est_un_entier(self):
    """ accesseur """
    ok = self.__valide
    ok = ok and (self.lire_num().degre() == 0)
    ok = ok and (self.lire_num().valuation().est_entier())
    ok = ok and (self.lire_denom().degre() == 0)
    ok = ok and (self.lire_denom().valuation().est_un())
    return ok

def simplifier_coefficients(self):
    """ simplifier les coefficients (numérateur et dénominateur) """
    if self.__valide:
        n = self.__num.pgcd_numerateurs()
        m = self.__denom.pgcd_numerateurs()
        d = ent.pgcd_entiers(n, m)
        r = ra.rationnel(d)

        p = po.polynome()
        for k in self.__num.liste_decroissante_monomes():
            c = k.lire_coeff()
            c /= r
            s = k.lire_indet()
            p = p.joinre(mo.monome(c, s))

        q = po.polynome()
        for k in self.__denom.liste_decroissante_monomes():
            c = k.lire_coeff()
            c /= r
            s = k.lire_indet()
            q = q.joinre(mo.monome(c, s))

        self.__num = p
        self.__denom = q

def reduire(self):
    """ reduction des coefficients """
    if self.__valide:
        n = self.__num.ppcm_denominateurs()
        m = self.__denom.ppcm_denominateurs()
        d = ent.pgcd_entiers(n, m)
        r = ra.rationnel((m * n) // d)

        p = po.polynome()

```

```

    for k in self.__num.liste_decroissante_monomes():
        c = k.lire_coeff()
        c *= r
        s = k.lire_indet()
        p = p.joinre(mo.monome(c, s))

    q = po.polynome()
    for k in self.__denom.liste_decroissante_monomes():
        c = k.lire_coeff()
        c *= r
        s = k.lire_indet()
        q = q.joinre(mo.monome(c, s))

    if q.degre() == 0:
        n = q.valuation().lire_num().lire_valeur()
        m = q.valuation().lire_denom().lire_valeur()
        r = ra.rationnel(m, n)
        t = po.polynome()
        for k in p.liste_decroissante_monomes():
            c = k.lire_coeff()
            c *= r
            s = k.lire_indet()
            t = t.joinre(mo.monome(c, s))
        p = t
        q = po.polynome_un()

    self.__num = p
    self.__denom = q

    self.simplifier_coefficients()

def fraction_err():
    """ fraction nulle obtenue par un calcul avec erreur """
    f = fraction()
    f.fixer_valide(False)
    return f

def fraction_nulle():
    """ fraction nulle """
    return fraction()

def fraction_un():
    """ fraction unite """
    return fraction(po.polynome().joinre(mo.monome(ra.rationnel(1))))

def fraction_depuis_lettre(lettre):
    """ construire une fraction rationnelle depuis une lettre """
    if lettre in string.ascii_letters:
        p = po.polynome()
        p = p.joinre(mo.monome(ra.rationnel(1), str(lettre)))
        return fraction(p, po.polynome_un())

    return fraction_err()

def fraction_depuis_naturel(n):
    """ construire une fraction rationnelle depuis un entier naturel """
    p = po.polynome()
    p = p.joinre(mo.monome(ra.rationnel(n)))

```

```
    return fraction(p, po.polynome_un())
```

```
if __name__ == "__main__":  
    pass
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

#
# fichier: fraction_tests.py
# version: 0.5.0
# auteur: Pascal CHAUVIN
# date: 2014/10/28
#
# (tous les symboles non internationaux sont volontairement omis)
#

import fraction as frac

import monome as mo
import polynome as po
import rationnel as ra

def test_unitaire_0(visible =False):
    print("*** fraction: test_unitaire_0 ***")

    f = frac.fraction()
    if visible: print(f)

    ok = f.est_valide()
    return ok

def test_unitaire_1(visible =False):
    print("*** fraction: test_unitaire_1 ***")

    f = frac.fraction(po.polynome_nul(), po.polynome_un(), False)
    if visible: print(f)

    ok = not f.est_valide()
    return ok

def test_unitaire_2(visible =False):
    print("*** fraction: test_unitaire_2 ***")

    a = po.polynome(mo.monome(ra.rationnel(1), "a"))
    b = po.polynome(mo.monome(ra.rationnel(1), "b"))

    c = po.polynome(mo.monome(ra.rationnel(1), "c"))
    d = po.polynome(mo.monome(ra.rationnel(1), "d"))

    f = frac.fraction(a, b)
    g = frac.fraction(c, d)

    r = f + g
    if visible:
        print(r)
        print(r.joli())

    ok = r.est_valide()
    return ok

def test_unitaire_3(visible =False):
    print("*** fraction: test_unitaire_3 ***")

    a = po.polynome(mo.monome(ra.rationnel(1), "a"))

    #
```

```
# on provoque volontairement une erreur
#
b = po.polynome(mo.monome(ra.rationnel(1), "b", False))

c = po.polynome(mo.monome(ra.rationnel(1), "c"))
d = po.polynome(mo.monome(ra.rationnel(1), "d"))

f = frac.fraction(a, b)
g = frac.fraction(c, d)

r = f + g
if visible: print(r)

ok = not r.est_valide()
return ok

def test_unitaire_4(visible =False):
    print("*** fraction: test_unitaire_4 ***")

    a = po.polynome(mo.monome(ra.rationnel(1), "x"))
    a = a.joindre(mo.monome(ra.rationnel(1)))
    # if visible: print(a)

    b = po.polynome(mo.monome(ra.rationnel(1), "x"))
    b = b.joindre(mo.monome(ra.rationnel(3)))
    # if visible: print(b)

    c = po.polynome(mo.monome(ra.rationnel(-1), "xx"))
    # if visible: print(c)

    d = po.polynome(mo.monome(ra.rationnel(1)))
    # if visible: print(d)

    f = frac.fraction(a, b)
    if visible: print(f)

    g = frac.fraction(c, d)
    if visible: print(g)

    r = f + g
    if visible: print(r)

    ok = r.est_valide()
    return ok

def test_unitaire_5(visible =False):
    print("*** fraction: test_unitaire_5 ***")

    a = po.polynome(mo.monome(ra.rationnel(1), "x"))
    a = a.joindre(mo.monome(ra.rationnel(1)))
    # if visible: print(a)

    b = po.polynome(mo.monome(ra.rationnel(1), "x"))
    b = b.joindre(mo.monome(ra.rationnel(3)))
    # if visible: print(b)

    c = po.polynome(mo.monome(ra.rationnel(-1), "xx"))
    # if visible: print(c)

    d = po.polynome(mo.monome(ra.rationnel(1)))
    # if visible: print(d)

    f = frac.fraction(a, b)
    # if visible: print(f)
```



```
g = frac.fraction(c, d)
# if visible: print(g)

r = f - g
if visible: print(r)

ok = r.est_valide()
return ok

def test_unitaire_6(visible =False):
    print("*** fraction: test_unitaire_6 ***")

    a = po.polynome(mo.monome(ra.rationnel(1), "x"))
    a = a.joindre(mo.monome(ra.rationnel(1)))
    # if visible: print(a)

    b = po.polynome(mo.monome(ra.rationnel(1), "x"))
    b = b.joindre(mo.monome(ra.rationnel(3)))
    # if visible: print(b)

    c = po.polynome(mo.monome(ra.rationnel(-1), "xx"))
    # if visible: print(c)

    d = po.polynome(mo.monome(ra.rationnel(1)))
    # if visible: print(d)

    f = frac.fraction(a, b)
    # if visible: print(f)

    g = frac.fraction(c, d)
    # if visible: print(g)

    r = f * g
    if visible: print(r)

    ok = r.est_valide()
    return ok

def test_unitaire_7(visible =False):
    print("*** fraction: test_unitaire_7 ***")

    a = po.polynome(mo.monome(ra.rationnel(1), "x"))
    a = a.joindre(mo.monome(ra.rationnel(1)))
    # if visible: print(a)

    b = po.polynome(mo.monome(ra.rationnel(1), "x"))
    b = b.joindre(mo.monome(ra.rationnel(3)))
    # if visible: print(b)

    c = po.polynome(mo.monome(ra.rationnel(-1), "xx"))
    # if visible: print(c)

    d = po.polynome(mo.monome(ra.rationnel(1)))
    # if visible: print(d)

    f = frac.fraction(a, b)
    # if visible: print(f)

    g = frac.fraction(c, d)
    # if visible: print(g)

    r = f / g
    if visible: print(r)
```

```
    ok = r.est_valide()
    return ok

def test_unitaire_8(visible =False):
    print("*** fraction: test_unitaire_8 ***")

    a = po.polynome(mo.monome(ra.rationnel(1), "a"))
    a = a.joindre(mo.monome(ra.rationnel(1), "b"))
    # if visible: print(a)

    b = po.polynome(mo.monome(ra.rationnel(3)))
    # if visible: print(b)

    c = po.polynome(mo.monome(ra.rationnel(5)))
    # if visible: print(c)

    f = frac.fraction(a, b)
    if visible: print(f)

    g = frac.fraction(c, po.polynome_un())
    if visible: print(g)

    r = f ** g
    if visible:
        print(r)
        print(r.joli())

    ok = (r.lire_num().valuation() == ra.rationnel(1, 243))
    return ok

def test_unitaire_9(visible =False):
    print("*** fraction: test_unitaire_9 ***")

    a = po.polynome(mo.monome(ra.rationnel(2)))
    # if visible: print(a)

    b = po.polynome(mo.monome(ra.rationnel(1)))
    # if visible: print(b)

    c = po.polynome(mo.monome(ra.rationnel(3)))
    # if visible: print(c)

    d = po.polynome(mo.monome(ra.rationnel(4)))
    # if visible: print(d)

    f = frac.fraction(a, b)
    # if visible: print(f)

    g = frac.fraction(c, d)
    # if visible: print(g)

    r = (f + g) + (f - g) + (f * g) + (f / g)
    if visible: print(r)

    ok = (r == ra.rationnel(49, 6))
    return ok

def test_unitaire_10(visible =False):
    print("*** fraction: test_unitaire_10 ***")

    a = po.polynome(mo.monome(ra.rationnel(1), "a"))
```

```
a = a.joindre(mo.monome(ra.rationnel(1), "b"))
# if visible: print(a)

b = po.polynome(mo.monome(ra.rationnel(3)))
# if visible: print(b)

f = frac.fraction(a, b)
if visible: print(f)

r = f ** 5
if visible: print(r)

ok = (r.lire_denom().valuation().lire_num().lire_valeur() == 243)
return ok
```

```
def test_unitaire_11(visible =False):
    print("*** fraction: test_unitaire_11 ***")

    f = frac.fraction_nulle()
    if visible: print("fraction nulle:", f)

    ok1 = f.lire_num().valuation().est_zero()

    f = frac.fraction_un()
    if visible: print("fraction unite:", f)
    ok2 = f.lire_num().valuation().est_un()

    f = frac.fraction_err()
    if visible: print("fraction erreur:", f)
    ok3 = not f.est_valide()

    ok = ok1 and ok2 and ok3
    return ok
```

```
def test_unitaire_12(visible =False):
    print("*** fraction: test_unitaire_12 ***")

    a = po.polynome(mo.monome(ra.rationnel(3)))
    if visible: print(a)

    b = po.polynome(mo.monome(ra.rationnel(19)))
    if visible: print(b)

    b = b.joindre(mo.monome(ra.rationnel(1)))
    if visible: print(b)

    b = b.joindre(mo.monome(ra.rationnel(-20)))
    if visible: print(b)

    f = frac.fraction(a, b)
    if visible: print(f)

    ok = (not f.est_valide())
    return ok
```

```
def test_unitaire_13(visible =False):
    print("*** fraction: test_unitaire_13 ***")

    ok = True
    return ok
```

```
def test_unitaire_14(visible =False):
    print("*** fraction: test_unitaire_14 ***")

    ok = True
    return ok
```

```
def test_unitaire_15(visible =False):
    print("*** fraction: test_unitaire_15 ***")

    ok = True
    return ok
```

```
def test_unitaire_16(visible =False):
    print("*** fraction: test_unitaire_16 ***")

    ok = True
    return ok
```

```
def test_unitaire_17(visible =False):
    print("*** fraction: test_unitaire_17 ***")

    ok = True
    return ok
```

```
def test_unitaire_18(visible =False):
    print("*** fraction: test_unitaire_18 ***")

    ok = True
    return ok
```

```
def test_unitaire_19(visible =False):
    print("*** fraction: test_unitaire_19 ***")

    ok = True
    return ok
```

```
def test_unitaire_(visible =False):
    print("*** fraction: test_unitaire_ ***")

    ok = True
    return ok
```

```
def tests_unitaires():
    return (
        test_unitaire_0() and \
        test_unitaire_1() and \
        test_unitaire_2(True) and \
        test_unitaire_3() and \
        test_unitaire_4(True) and \
        test_unitaire_5(True) and \
        test_unitaire_6(True) and \
        test_unitaire_7(True) and \
        test_unitaire_8(True) and \
```

```
test_unitaire_9() and \
test_unitaire_10() and \
test_unitaire_11(True) and \
test_unitaire_12(True) and \
test_unitaire_13() and \
test_unitaire_14() and \
test_unitaire_15() and \
test_unitaire_16() and \
test_unitaire_17() and \
test_unitaire_18() and \
test_unitaire_19()
)

if __name__ == "__main__":
    ok = tests_unitaires()
    if ok:
        print("*** fraction: tests unitaires OK ***")
```

