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

#
# fichier: fraction.py
#    date: 2011/05/04
#
# (tous les symboles non internationaux sont volontairement omis)
#

import string

from polynome import *

class fraction(object):

  def __init__(self, num =polynome_nul(), denom =polynome_un(), valide =True):
    """ constructeur """
    self.__valide = valide and num.est_valide() and denom.est_valide() and (not denom.est_n
ul())
    if self.__valide:
      self.__num = num
      self.__denom = denom
      self.reduire()
    else:
      self.__num = polynome_nul()
      self.__denom = polynome_un()



  def __str__(self):
    """ representation en chaine de caracteres """
    if self.__valide:
      if self.__denom.est_unite():
        return str(self.__num)
      else:
        a = str(self.__num)
        if self.__num.nombre_monomes() == 1:
          while a.startswith("(") and a.endswith(")"):
            a = a[1:-1]
        b = str(self.__denom)
        if self.__denom.nombre_monomes() == 1:
          while b.startswith("(") and b.endswith(")"):
            b = b[1:-1]
        return "(" + a + ")/(" + b + ")"
    else:
      return "(fraction invalide)"



  def simplifier_coefficients(self):
    """ simplification des coefficients du numerateur et du denominateur """
    if self.__valide:
      n = self.__num.pgcd_numerateurs()
      m = self.__denom.pgcd_numerateurs()
      d = pgcd_naturel(n, m)
      r = rationnel(d)

      p = polynome()
      for k in self.__num.liste_decroissante():
        c = k.get_coefficient()
        s = k.get_indeterminee()
        c /= r
        p.ajouter_monome(monome(c, s))

      q = polynome()
      for k in self.__denom.liste_decroissante():
        c = k.get_coefficient()
```

```
      s = k.get_indeterminee()
      c /= r
      q.ajouter_monome(monome(c, s))

    self.__num = p
    self.__denom = q



  def reduire(self):
    """ reduction des coefficients du numerateur et du denominateur """
    if self.__valide:
      n = self.__num.ppcm_denominateurs()
      m = self.__denom.ppcm_denominateurs()
      d = pgcd_naturel(n, m)
      r = rationnel(m*n/d)

      p = polynome()
      for k in self.__num.liste_decroissante():
        c = k.get_coefficient()
        s = k.get_indeterminee()
        c *= r
        p.ajouter_monome(monome(c, s))

      q = polynome()
      for k in self.__denom.liste_decroissante():
        c = k.get_coefficient()
        s = k.get_indeterminee()
        c *= r
        q.ajouter_monome(monome(c, s))

      if q.degre() == 0:
        n = q.valuation().get_num().valeur()
        m = q.valuation().get_denom().valeur()
        r = rationnel(m, n)
        t = polynome()
        for k in p.liste_decroissante():
          c = k.get_coefficient()
          s = k.get_indeterminee()
          c *= r
          t.ajouter_monome(monome(c, s))
        p = t
        q = polynome_un()

      self.__num = p
      self.__denom = q

      self.simplifier_coefficients()



  def est_valide(self):
    """ indique l'etat de validite """
    return self.__valide



  def valider(self):
    """ valider l'objet """
    self.__valide = True



  def invalider(self):
    """ invalider l'objet """
    self.__valide = False
```

```
  def __add__(self, autre):
    """ addition """
    if self.__valide and autre.__valide:
      a, b = self.__num, self.__denom
      p, q = autre.__num, autre.__denom
      return fraction(a*q + b*p, b*q)
    else:
      return fraction(polynome_nul(), polynome_un(), False)



  def __neg__(self):
    """ fraction opposee """
    if self.__valide:
      a, b = self.__num, self.__denom
      return fraction(-a, b)
    else:
      return fraction(polynome_nul(), polynome_un(), False)



  def __sub__(self, autre):
    """ addition de la fraction opposee """
    return self.__add__(autre.__neg__())



  def __mul__(self, autre):
    """ multiplication """
    if self.__valide and autre.__valide:
      a, b = self.__num, self.__denom
      p, q = autre.__num, autre.__denom
      return fraction(a*p, b*q)
    else:
      return fraction(polynome_nul(), polynome_un(), False)



  def __div__(self, autre):
    """ division """
    if self.__valide and autre.__valide:
      a, b = self.__num, self.__denom
      p, q = autre.__num, autre.__denom
      return fraction(a*q, b*p)
    else:
      return fraction(polynome_nul(), polynome_un(), False)



  def __pow__(self, autre):
    """ exponentiation (si exposant entier relatif) """
    if self.__valide and autre.__valide:
      a, b = self.__num, self.__denom
      p, q = autre.__num, autre.__denom

      if (p.degre() != 0) or (q.degre() != 0):
        return fraction(polynome_nul(), polynome_un(), False)

      r = p.valuation() / q.valuation()
      if r.get_num().valeur() < 0:
        r, a, b = -r, b, a

      if r.est_entier():
        t = polynome()
        t.ajouter_monome(monome(r))
        return fraction(a**t, b**t)
```

```
    return fraction(polynome_nul(), polynome_un(), False)



def fraction_depuis_lettre(c):
  """ construire une fraction (rationnelle) depuis une lettre """
  if c in string.letters:
    p = polynome()
    p.ajouter_monome(monome(rationnel(1), str(c)))
    return fraction(p, polynome_un())
  else:
    return fraction(polynome_nul(), polynome_un(), False)



def fraction_depuis_naturel(x):
  """ construire une fraction (rationnelle) depuis un entier naturel """
  p = polynome()
  p.ajouter_monome(monome(rationnel(x)))
  return fraction(p, polynome_un())



def fraction_nulle_erronee():
  """ fraction nulle (erreur) """
  return fraction(polynome_nul(), polynome_un(), False)



#
# tests unitaires
#
def test_01():
  print "\n*** test 1 ***"
  x = fraction()
  print x
  print x.est_valide()



def test_02():
  print "\n*** test 2 ***"
  a = polynome()
  a.ajouter_monome(monome(rationnel(1), "x"))
  a.ajouter_monome(monome(rationnel(2)))

  b = polynome()
  b.ajouter_monome(monome(rationnel(1), "xx"))
  b.ajouter_monome(monome(rationnel(2, 3)))

  u = fraction(a, b)
  print u

  c = polynome()
  c.ajouter_monome(monome(rationnel(1), "x"))
  c.ajouter_monome(monome(rationnel(1)))

  d = polynome()
  d.ajouter_monome(monome(rationnel(1), "x"))
  d.ajouter_monome(monome(rationnel(-1)))

  v = fraction(c, d)
  print v

  x = u + v
  print x
```

```
def test_03():
  print "\n*** test 3 ***"
  a = polynome()
  a.ajouter_monome(monome(rationnel(2, 3), "x"))
  a.ajouter_monome(monome(rationnel(1, 7)))

  b = polynome()
  b.ajouter_monome(monome(rationnel(-1, 24)))

  u = fraction(a, b)
  print u



def test_04():
  print "\n*** test 4 ***"
  a = polynome()
  a.ajouter_monome(monome(rationnel(2, 3), "x"))
  a.ajouter_monome(monome(rationnel(1, 7)))

  b = polynome_un()

  u = fraction(a, b)
  print u



def test_05():
  print "\n*** test 5 ***"
  a = polynome()
  a.ajouter_monome(monome(rationnel(2, 3), "x"))
  a.ajouter_monome(monome(rationnel(1, 7)))

  b = polynome_un()

  b = polynome()
  b.ajouter_monome(monome(rationnel(4, 15), "y"))
  b.ajouter_monome(monome(rationnel(-1, 24)))

  u = fraction(a, b)
  print u

  c = polynome()
  c.ajouter_monome(monome(rationnel(1), "x"))
  c.ajouter_monome(monome(rationnel(1)))

  d = polynome()
  d.ajouter_monome(monome(rationnel(1), "x"))
  d.ajouter_monome(monome(rationnel(-1)))

  v = fraction(c, d)
  print v

  x = u + v
  print x

  x = u - v
  print x

  x = u * v
  print x

  x = u / v
  print x
```

```
def test_06():
  print "\n*** test 6 ***"
  a = polynome()
  a.ajouter_monome(monome(rationnel(1), "x"))
  a.ajouter_monome(monome(rationnel(-1)))

  b = polynome_un()

  b = polynome()
  b.ajouter_monome(monome(rationnel(1), "y"))
  b.ajouter_monome(monome(rationnel(2)))

  u = fraction(a, b)
  print u

  e = polynome()
  e.ajouter_monome(monome(rationnel(-8, -2)))
  n = fraction(e, polynome_un())
  print n

  x = u ** n
  print x



def test_07():
  print "\n*** test 7 ***"
  f = fraction_depuis_lettre("x")
  print f

  g = fraction_depuis_naturel(3)
  print g

  s = f + g
  print s



def test_08():
  print "\n*** test 8 ***"
  a = polynome()
  a.ajouter_monome(monome(rationnel(1), "x"))
  a.ajouter_monome(monome(rationnel(-1)))

  b = polynome()
  b.ajouter_monome(monome(rationnel(1), "y"))
  b.ajouter_monome(monome(rationnel(2)))

  u = fraction(a, b)
  print u

  c = polynome()
  c.ajouter_monome(monome(rationnel(4), "x"))

  d = polynome()
  d.ajouter_monome(monome(rationnel(1), "x"))

  v = fraction(c, d)
  print v

  x = u ** v
  print x



def test_09():
  print "\n*** test 9 ***"
  a = polynome()
```

```
  a.ajouter_monome(monome(rationnel(1), "x"))
  a.ajouter_monome(monome(rationnel(-1)))

  b = polynome()
  b.ajouter_monome(monome(rationnel(1), "y"))
  b.ajouter_monome(monome(rationnel(2)))

  u = fraction(a, b)
  print u

  c = polynome()
  c.ajouter_monome(monome(rationnel(4)))

  d = polynome()
  d.ajouter_monome(monome(rationnel(1)))

  v = fraction(c, d)
  print v

  x = u ** v
  print x



def test_10():
  print "\n*** test 10 ***"
  a = polynome()
  a.ajouter_monome(monome(rationnel(6), "x"))
  a.ajouter_monome(monome(rationnel(-24)))

  b = polynome()
  b.ajouter_monome(monome(rationnel(12), "y"))
  b.ajouter_monome(monome(rationnel(-6)))

  u = fraction(a, b)
  print u



def test_11():
  print "\n*** test 11 ***"
  a = polynome()
# a.ajouter_monome(monome(rationnel(6), "x"))
# a.ajouter_monome(monome(rationnel(-24)))
  a.ajouter_monome(monome(rationnel(-24), "x"))

  b = polynome()
  b.ajouter_monome(monome(rationnel(12), "y"))
  b.ajouter_monome(monome(rationnel(-6)))

  u = fraction(a, b)
  print u



def test_12():
  print "\n*** test 12 ***"
  a = polynome()
  a.ajouter_monome(monome(rationnel(2, 3), "x"))
  a.ajouter_monome(monome(rationnel(1, 7)))

  b = polynome()
  b.ajouter_monome(monome(rationnel(4, 15), "y"))
  b.ajouter_monome(monome(rationnel(-1, 24)))

  u = fraction(a, b)
  print u
```

```
  c = polynome()
  c.ajouter_monome(monome(rationnel(4)))

  d = polynome_un()

  v = fraction(c, d)
  print v

  x = u ** v
  print x



def tests_unitaires():
  test_01()
  test_02()
  test_03()
  test_04()
  test_05()
  test_06()
  test_07()
  test_08()
  test_09()
  test_10()
  test_11()
  test_12()



if __name__ == "__main__":
  tests_unitaires()
```