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

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

from entier import *

def produit_2_5(n):
  """ n est-il de la forme (2^p)*(5^q) ? """
  if n == 1:
    return True
  else:
    if n % 2 == 0:
      return produit_2_5(n/2)
    if n % 5 == 0:
      return produit_2_5(n/5)
    return False



class rationnel(object):
  """ classe pour un nombre rationnel """

  def __init__(self, num =long(0), denom =long(1), valide =True):
    """ constructeur """
    valide = valide and (type(num) == int or type(num) == long or isinstance(num, entier))
    valide = valide and (type(denom) == int or type(denom) == long or isinstance(denom, ent
ier))
    if valide:
      if type(num) == int or type(num) == long:
        num = entier(num)
      if type(denom) == int or type(denom) == long:
        denom = entier(denom)
    valide = valide and num.est_valide() and denom.est_valide()
    self.__valide = valide and (denom.valeur() != 0)
    if self.__valide:
      if denom.valeur() < 0:
        num, denom = -num, -denom
      d = pgcd_entier(num, denom)
      self.__num, self.__denom = num / d, denom / d
    else:
      self.__num, self.__denom = entier(0), entier(1)



  def __str__(self):
    """ representation en chaine de caracteres """
    if self.__valide:
      if self.__denom.valeur() == 1:
        return str(self.__num)
      if produit_2_5(self.__denom.valeur()):
        return str(self.__float__())
      else:
        return str(self.__num) + "/" + str(self.__denom)
    return "(nombre rationnel invalide)"



  def __float__(self):
    """ donne (si possible) la representation decimale, sinon 0.0 """
    if self.__valide and produit_2_5(self.__denom.valeur()):
      a, b = self.__num.valeur(), self.__denom.valeur()
      return (float(a)/float(b))
```

```
    return float(0)


  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 rationnel(a*q + b*p, b*q)
    else:
      return rationnel(0, 1, False)


  def __neg__(self):
    """ rationnel oppose """
    if self.__valide:
      a, b = self.__num, self.__denom
      return rationnel(-a, b)
    else:
      return rationnel(0, 1, False)


  def __sub__(self, autre):
    """ addition de l'oppose """
    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 rationnel(a*p, b*q)
    else:
      return rationnel(0, 1, 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 rationnel(a*q, b*p)
    else:
      return rationnel(0, 1, 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 (a.valeur() == 0) and (p.valeur() <= 0):
        return rationnel(0, 1, False)
      if p.valeur() < 0:
        p, a, b = -p, b, a
      if b.valeur() < 0:
        a, b = -a, -b
      if q.valeur() == 1:
        return rationnel(a**p, b**p)
    return rationnel(0, 1, False)



  def est_nul(self):
    """ le rationnel est-il nul ? """
    if self.__valide:
      return self.__num.valeur() == 0
    else:
      return False



  def est_positif(self):
    """ le rationnel est-il positif ? """
    if self.__valide:
      return self.__num.est_positif()
    else:
      return False



  def est_entier(self):
    """ le rationnel est-il entier ? """
    if self.__valide:
      return (self.__denom.valeur() == 1)
    else:
      return False



  def est_unite(self):
    """ le polynome est-il egal a 1 ? """
    if self.__valide:
      return (self.__num.valeur() == self.__denom.valeur())
    else:
      return False



  def get_num(self):
    """ accesseur du numerateur """
    return self.__num



  def get_denom(self):
    """ accesseur du denominateur """
    return self.__denom
```

```
if __name__ == "__main__":

  x = rationnel(-10, -40)
  print x.est_valide()
  print x
  print float(x)

  y = rationnel(-10, -41)
  print y
  print float(y)

  z = rationnel(10, -45)
  print z.est_valide()
  print z

  z = rationnel(1, 560)
  n = rationnel(-4)

  p = z ** n
  print p.est_valide()
  print p
```