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

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

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

import entier as ent
import monome as mo
import joli
import rationnel as ra
import utile

class polynome(object):

  def __init__(self, monome =mo.monome(), gauche =None, droite =None, valide =True):
    """ - """
    if valide:
      self.__monome = monome
    else:
      self.__monome = None

    self.__valide = valide and monome.est_valide()
    self.__gauche = gauche
    self.__droite = droite



  def plat(self):
    """ - """
    acc = ""
    if self.__droite is not None:
      acc = " + " + self.__droite.plat()

    acc = str(self.__monome) + acc

    if self.__gauche is not None:
      acc = self.__gauche.plat() + " + " + acc

    return acc



  def __repr__(self):
    """ - """
    gauche = None
    if self.__gauche is not None:
      gauche = id(self.__gauche)

    droite = None
    if self.__droite is not None:
      droite = id(self.__droite)

    return "[polynome:\n__monome={},\n__gauche={},\n__droite={},\n__valide={}\n]\n"\
      .format(self.__monome, gauche, droite, self.__valide)
```

```
  def __str__(self):
    return self.plat()



  def joli(self):
    """ - """
    acc = ""
    if self.__droite is not None:
      acc = " + " + self.__droite.joli()

    acc = self.__monome.joli() + acc

    if self.__gauche is not None:
      acc = self.__gauche.joli() + " + " + acc

    return acc



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



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



  def lire_monome(self):
    return self.__monome



  def __inserer(self, k):
    if self.__monome < k:
      if self.__gauche is None:
        self.__gauche = polynome(k)
      else:
        self.__gauche.__inserer(k)
      return

    if self.__droite is None:
      self.__droite = polynome(k)
    else:
      self.__droite.__inserer(k)



  def inserer(self, k):
    if self.__monome.lire_coeff().est_zero():
      self.__monome = k
    else:
      self.__inserer(k)



  def __iterateur(self, acc):
    """ - """
    if self.__gauche is not None:
       self.__gauche.__iterateur(acc)

    acc.append(self.__monome)
```

```
    if self.__droite is not None:
      self.__droite.__iterateur(acc)



  def iterateur(self):
    """ - """
    acc = []
    self.__iterateur(acc)
    return acc



  def joindre(self, k):
    """ ajouter un monome donne un nouveau polynome """
    if not k.est_valide:
      p = polynome()
      p.fixer_valide(False)
      return p

    if k.lire_coeff().est_zero():
      p = polynome()
      u = self.iterateur()
      for i in u:
        p.inserer(i)
      return p

    p = polynome()
    u = self.iterateur()
    trouve = False

    for i in u:
      if i.lire_indet() == k.lire_indet():
        trouve = True
        c = i.lire_coeff() + k.lire_coeff()
        if not c.est_zero():
          p.inserer(mo.monome(c, k.lire_indet()))
      else:
        p.inserer(i)

    if not trouve:
      p.inserer(k)

    return p



  def contient(self, k):
    """ - """
    if self.__monome == k:
      return True

    if self.__monome < k:
      if self.__gauche is None:
        return False
      else:
        return self.__gauche.contient(k)

    if self.__droite is None:
      return False
    else:
      return self.__droite.contient(k)



  def debut(self):
    """ debut = monome le plus a gauche """
```

```
    if self.__gauche is None:
      return self
    else:
      return self.__gauche.debut()



  def degre(self):
    """ donne le degre du polynome """
    m = self.debut().lire_monome()
    if m is None:
      return (-1) # concession a la def. math. du degre du pol. nul

    if m.est_degre_nul():
      return 0

    return len(m.lire_indet())



  def fin(self):
    """ fin = monome le plus a droite """
    if self.__droite is None:
      return self
    else:
      return self.__droite.fin()



  def valuation(self):
    """ - """
    m = self.fin().lire_monome()
    if m:
      return m.lire_coeff()

    return rat.rationnel()



  def nombre_monomes(self):
    """ nombre de monomes """
    n = 1

    if self.__gauche is not None:
      n += self.__gauche.nombre_monomes()

    if self.__droite is not None:
      n += self.__droite.nombre_monomes()

    return n



  def __add__(self, autre):
    """ addition """
    if isinstance(autre, polynome):
      if (self.__valide) and (autre.__valide):
        p = polynome()

        for m in self.iterateur():
          p = p.joindre(m)

        for m in autre.iterateur():
          p = p.joindre(m)

        return p

    return polynome(mo.monome(), None, None, False)
```

```
def __neg__(self):
  """ polynome oppose (inverse pour l'addition) """
  if self.__valide:
    p = polynome()

    for m in self.iterateur():
      t = mo.monome(-m.lire_coeff(), m.lire_indet(), m.est_valide())
      p = p.joindre(t)

    return p

  return polynome(mo.monome(), None, None, False)



def oppose(self):
  """ polynome oppose """
  return self.__neg__()



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



def __mul__(self, autre):
  """ produit de deux polynomes """
  if isinstance(autre, polynome):
    if (self.__valide) and (autre.__valide):
      p = polynome()

      for m in self.iterateur():
        m_coeff = m.lire_coeff()
        m_indet = m.lire_indet()

        for n in autre.iterateur():
          n_coeff = n.lire_coeff()
          n_indet = n.lire_indet()

          i = utile.reduction(m_indet + n_indet)

          k = mo.monome(m_coeff * n_coeff, i)
          p = p.joindre(k)

      return p

  return polynome(mo.monome(), None, None, False)



def __exponentiation(self, n):
  """ exponentiation (exposant entier naturel) """
  p = polynome()

  a = self
  p = p.joindre(mo.monome(ra.rationnel(1)))

  while n > 0:
    if n % 2 == 1:
      p *= a
    n //= 2
    a *= a
```

```
    return p


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

    if isinstance(autre, polynome):
      if (self.__valide) and (autre.__valide):
        v = autre.valuation()

        if self.est_polynome_nul() and v.lire_num().est_zero():
          return polynome(mo.monome(), None, None, False)

        n = v.lire_num().lire_valeur()

        if (self.degre() != 0) and (n < 0):
          return polynome(mo.monome(), None, None, False)

        return self.__exponentiation(n)

    return polynome(mo.monome(), None, None, False)


  def est_degre_nul(self):
    """ - """
    return (self.degre() == 0)


  def est_polynome_nul(self):
    """ - """
    return self.valuation().lire_num().est_zero()


  def est_polynome_unite(self):
    """ - """
    return (self.degre() == 0) and \
      self.valuation().lire_num().est_un()


  def liste_decroissante_monomes(self):
    """ - """
    liste = []

    it = self.iterateur()
    for mono in it:
      liste.append(mono) # i.e. liste.append(repr(mono))

    return sorted(liste, reverse = True)


  def pgcd_numerateurs(self):
    """ - """
    l = []
    if self.__valide:
      for m in self.liste_decroissante_monomes():
        e = m.lire_coeff().lire_num().lire_valeur()
        if not (e in l):
          l.append(e)
    return ent.pgcd_liste(l)
```

```
  def ppcm_denominateurs(self):
    """ - """
    l = []
    n = 1
    if self.__valide:
      for m in self.liste_decroissante_monomes():
        """ les denominateurs sont positifs """
        e = m.lire_coeff().lire_denom().lire_valeur()
        if not (e in l):
          l.append(e)
        n *= e
    return (n // ent.pgcd_liste(l))



def polynome_err():
  """ polynome nul obtenu par un calcul avec erreur """
  p = polynome()
  p.fixer_valide(False)
  return p



def polynome_nul():
  """ polynome nul """
  return polynome()



def polynome_un():
  """ polynome unite """
  return polynome().joindre(mo.monome(ra.rationnel(1)))



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

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

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

import sys
sys.path.append('../monome_mod')
sys.path.append('../rationnel_mod')

import polynome as po

import monome as mo
import rationnel as ra

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

  a = po.polynome()
  if visible: print(a)

  ok = (a.nombre_monomes() == 1)
  return ok



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

  a = po.polynome(mo.monome(ra.rationnel(12), "x"))
  if visible:
    print(a)
    print(repr(a))
    print(a.lire_monome())
    print(a.plat())

  ok = a.contient(mo.monome(ra.rationnel(1), "x"))
  return ok



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

  a = po.polynome(mo.monome(ra.rationnel(12), "x"))
  if visible: print(a.plat())

  a.inserer(mo.monome(ra.rationnel(1), "y"))
  if visible: print(a.plat())

  a.inserer(mo.monome(ra.rationnel(1), "a"))
  if visible: print(a.plat())

  if visible: print("---")

  b = po.polynome()
  for t in a.iterateur():
    b.inserer(t)

  if visible: print(b.plat())

  ok = (a.fin().lire_monome().lire_indet() == "y")
```

```
  return ok


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

  a = po.polynome()
  if visible: print(a.plat())

  a.inserer(mo.monome(ra.rationnel(12), "x"))
  if visible: print(a.plat())

  a.inserer(mo.monome(ra.rationnel(1), "y"))
  if visible: print(a.plat())

  a.inserer(mo.monome(ra.rationnel(1), "a"))
  if visible: print(a.plat())

  if visible: print("---")

  b = po.polynome()
  for t in a.iterateur():
    b.inserer(t)

  if visible: print(b.plat())

  ok = (a.nombre_monomes() == 3)
  return ok



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

  a = po.polynome()
  if visible: print(a.plat())

  a.inserer(mo.monome(ra.rationnel(12), "x"))
  if visible: print(a.plat())

  a.inserer(mo.monome(ra.rationnel(1), "y"))
  if visible: print(a.plat())

  a.inserer(mo.monome(ra.rationnel(1), "a"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(100), "t"))
  if visible: print(a.plat())

  ok = (a.debut().lire_monome().lire_coeff() == ra.rationnel(1)) and \
    (a.nombre_monomes() == 4)
  return ok



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

  a = po.polynome()
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(12), "x"))
  if visible: print(a.plat())

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

```
  a = a.joindre(mo.monome(ra.rationnel(-12), "x"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(100), "t"))
  if visible: print(a.plat())

# ok = (a.debut().lire_monome().lire_coeff().est_un())
#   (a.nombre_monomes() == 2)
  ok = (a.nombre_monomes() == 2) and \
    (a.debut().lire_monome().lire_coeff().lire_num().lire_valeur() == 100)
  return ok



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

  a = po.polynome()
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(12), "x"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(1), "y"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(), "y"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(-1), "y"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(100), "t"))
  if visible: print(a.plat())

  ok = (a.nombre_monomes() == 2) and \
    (a.debut().lire_monome().lire_coeff().lire_num().lire_valeur() == 100)
  return ok



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

  a = po.polynome()
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(12), "x"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(1), "y"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(5), "x"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(100), "t"))
  if visible: print(a.plat())

  ok = a.contient(mo.monome(ra.rationnel(1), "x"))
  return ok



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

  a = po.polynome()
```

```
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(12), "x"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(1), "y"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(5), "x"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(100), "t"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(-12), "x"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(-1), "y"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(-5), "x"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(-100), "t"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(0), "t"))
  if visible: print(a.plat())

  a = a.joindre(mo.monome(ra.rationnel(3, -30), "xx"))
  if visible: print(a.plat())

  ok = (a.nombre_monomes() == 1) and \
    (a.debut().lire_monome().lire_coeff() == ra.rationnel(-1, 10))
  return ok



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

  p = po.polynome()
  if visible: print(p)

  p = p.joindre(mo.monome(ra.rationnel(2), "x"))
  if visible: print(p)

  p = p.joindre(mo.monome(ra.rationnel(-2,3), "x"))
  if visible: print(p)

  ok = (p.nombre_monomes() == 1) and \
    (p.lire_monome().lire_coeff() == ra.rationnel(4, 3)) and \
    (p.lire_monome().lire_indet() == "x")
  return ok



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

  p = po.polynome()
  p = p.joindre(mo.monome(ra.rationnel(2), "x"))
  p = p.joindre(mo.monome(ra.rationnel(1), "y"))
  p = p.joindre(mo.monome(ra.rationnel(2), "x"))
  p = p.joindre(mo.monome(ra.rationnel(-3), "y"))

  if visible: print(p)
```

```
  v1 = p.fin().lire_monome().lire_coeff().lire_num().lire_valeur()
  v2 = p.debut().lire_monome().lire_coeff().lire_num().lire_valeur()

  ok = (v1 == -2) and (v2 == 4)
  return ok



def test_unitaire_11(visible =False):
  print("*** polynome: test_unitaire_11 ***")

  p = po.polynome()
  p = p.joindre(mo.monome(ra.rationnel(2), "x"))
  p = p.joindre(mo.monome(ra.rationnel(1), "y"))

  if visible: print(p)

  q = po.polynome()
  q = q.joindre(mo.monome(ra.rationnel(-5), "a"))
  q = q.joindre(mo.monome(ra.rationnel(10), "y"))

  if visible: print(q)

  r = p + q

  if visible: print(r)

  ok = (r.valuation().lire_num().lire_valeur() == 11)
  return ok



def test_unitaire_12(visible =False):
  print("*** polynome: test_unitaire_12 ***")

  p = po.polynome()
  p = p.joindre(mo.monome(ra.rationnel(2), "x"))
  p = p.joindre(mo.monome(ra.rationnel(1), "y"))

  if visible:
    print(p)
    print(-p)

  q = po.polynome()
  q = q.joindre(mo.monome(ra.rationnel(-5), "a"))
  q = q.joindre(mo.monome(ra.rationnel(10), "y"))

  if visible: print(q)

  r = p - q

  if visible: print(r)

  ok = (r.valuation().lire_num().lire_valeur() == -9)
  return ok



def test_unitaire_13(visible =False):
  print("*** polynome: test_unitaire_13 ***")

  p = po.polynome(mo.monome(ra.rationnel(20, -30), "xax"))
  if visible: print(p)

  p = p.joindre(mo.monome(ra.rationnel(1), "x"))
  if visible: print(p)

  p = p.joindre(mo.monome(ra.rationnel(1), "a"))
```

```
  if visible: print(p)

  p = p.joindre(mo.monome(ra.rationnel(-1), "x"))
  if visible: print(p)

  ok = (p.degre() == 3) and (p.valuation() == ra.rationnel(1))
  return ok



def test_unitaire_14(visible =False):
  print("*** polynome: test_unitaire_14 ***")

  p = po.polynome_nul()
  if visible: print("Polynome nul:", p)

  ok1 = p.valuation().est_zero()

  p = po.polynome_un()
  if visible: print("Polynome unite:", p)
  ok2 = p.valuation().est_un()

  p = po.polynome_err()
  if visible: print("Polynome erreur:", p)
  ok3 = not p.est_valide()

  ok = ok1 and ok2 and ok3
  return ok



def test_unitaire_15(visible =False):
  print("*** polynome: test_unitaire_15 ***")

  p = po.polynome()
  p = p.joindre(mo.monome(ra.rationnel(2), "x"))
  p = p.joindre(mo.monome(ra.rationnel(1), "y"))

  if visible: print(p)

  q = po.polynome()
  q = q.joindre(mo.monome(ra.rationnel(-5), "a"))
  q = q.joindre(mo.monome(ra.rationnel(10), "y"))

  if visible: print(q)

  r = p * q

  if visible: print(r)

  ok = (r.valuation() == ra.rationnel(10))
  return ok



def test_unitaire_16(visible =False):
  print("*** polynome: test_unitaire_16 ***")

  p = po.polynome(mo.monome(ra.rationnel(-5, 2), "a"))
  if visible: print(p)

  p = p.joindre(mo.monome(ra.rationnel(15, 4)))
  if visible: print(p)

  p = p.joindre(mo.monome(ra.rationnel(5), "x"))
  if visible: print(p)

  if visible: print(p.valuation())
```

```
  ok = (p.valuation() == ra.rationnel(15, 4))
  return ok



def test_unitaire_17(visible =False):
  print("*** polynome: test_unitaire_17 ***")

  p = po.polynome()
  p = p.joindre(mo.monome(ra.rationnel(2, 3), "a"))
  p = p.joindre(mo.monome(ra.rationnel(1), "b"))

  if visible: print(p)

  q = po.polynome()
  q = q.joindre(mo.monome(ra.rationnel(2, 3), "a"))
  q = q.joindre(mo.monome(ra.rationnel(1), "b"))

  if visible: print(q)

  r = p * q

  if visible: print(r)

  if visible: print(r.valuation())

  ok = (r.valuation() == ra.rationnel(1))
  return ok



def test_unitaire_18(visible =False):
  print("*** polynome: test_unitaire_18 ***")

  p = po.polynome()
  p = p.joindre(mo.monome(ra.rationnel(2), "x"))
  p = p.joindre(mo.monome(ra.rationnel(-1)))

  if visible: print(p)

  q = po.polynome()
  q = q.joindre(mo.monome(ra.rationnel(2), "x"))
  q = q.joindre(mo.monome(ra.rationnel(1)))

  if visible: print(q)

  r = p * q

  if visible: print(r)

  if visible: print(r.valuation())

  ok = (r.valuation() == ra.rationnel(-1))
  return ok



def test_unitaire_19(visible =False):
  print("*** polynome: test_unitaire_19 ***")

  p = po.polynome()

  p = p.joindre(mo.monome(ra.rationnel(5, 7)))

  if visible:
    print(p)
    print(p.degre())
```

```
    print(p.valuation())

  ok = (p.degre() == 0) and (p.valuation() == ra.rationnel(5, 7))
  return ok



def test_unitaire_20(visible =False):
  print("*** polynome: test_unitaire_20 ***")

  p = po.polynome()

  p = p.joindre(mo.monome(ra.rationnel(2, 3), "xxxx"))
  p = p.joindre(mo.monome(ra.rationnel(1), "xxxyyy"))
  p = p.joindre(mo.monome(ra.rationnel(5, 3)))
  p = p.joindre(mo.monome(ra.rationnel(-5, 2), "a"))

  if visible:
    print(p)
    print(p.degre())
    print(p.valuation())

  ok = (p.degre() == 6) and (p.valuation() == ra.rationnel(5, 3))
  return ok



def test_unitaire_21(visible =False):
  print("*** polynome: test_unitaire_21 ***")

  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)

  ok = (b.valuation() == ra.rationnel(3))
  return ok



def test_unitaire_22(visible =False):
  print("*** polynome: test_unitaire_22 ***")

  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)

  r = (a*d)
  if visible: print(r)

  r = (b*c)
  if visible: print(r)

  r = (a*d + b*c)
  if visible: print(r)
```

```
  r = (b * d)
  if visible: print(r)

  ok = (r.valuation() == ra.rationnel(3))
  return ok



def test_unitaire_23(visible =False):
  print("*** polynome: test_unitaire_23 ***")

  p = po.polynome()
  p = p.joindre(mo.monome(ra.rationnel(1), "x"))
  p = p.joindre(mo.monome(ra.rationnel(1)))

  if visible: print(p)

  q = po.polynome()
  q = q.joindre(mo.monome(ra.rationnel(1), "x"))
  q = q.joindre(mo.monome(ra.rationnel(3)))

  if visible: print(q)

  r = p * q

  if visible:
    print(r)
    print(repr(r))

  ok = (r.degre() == 2)
  return ok



def test_unitaire_24(visible =False):
  print("*** polynome: test_unitaire_24 ***")

  p = po.polynome()
  ok1 = p.est_polynome_nul()

  p = p.joindre(mo.monome(ra.rationnel(8, -2)))
  ok2 = p.est_degre_nul()

  ok = ok1 and ok2
  return ok



def test_unitaire_25(visible =False):
  print("*** polynome: test_unitaire_25 ***")

  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(7)))
  if visible: print(b)

  r = a ** b
  if visible: print(r)

  ok = (r.valuation() == ra.rationnel(1))
  return ok



def test_unitaire_26(visible =False):
```

```
  print("*** polynome: test_unitaire_26 ***")

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

  r = a ** 3
  if visible: print(r)

  ok = (r.valuation() == ra.rationnel(1))

  return ok



def test_unitaire_27(visible =False):
  print("*** polynome: test_unitaire_27 ***")

  a = po.polynome(mo.monome(ra.rationnel(1), "a"))
  a = a.joindre(mo.monome(ra.rationnel(1), "b"))
  a = a.joindre(mo.monome(ra.rationnel(-2, 3)))
  if visible: print(a)

  b = po.polynome(mo.monome(ra.rationnel(7)))
  if visible: print(b)

  r = a ** b

  t = r.liste_decroissante_monomes()
  c = t[-1].lire_coeff()

  if visible:
    print(r)

    print(t)
    print(c)

    print(r.pgcd_numerateurs())
    print(r.ppcm_denominateurs())

  ok = (c == ra.rationnel(-128, 2187))
  return ok



def test_unitaire_28(visible =False):
  print("*** polynome: test_unitaire_28 ***")

  a = po.polynome(mo.monome(ra.rationnel(1), "x"))
  a = a.joindre(mo.monome(ra.rationnel(-2, 3)))
# if visible: print(a)

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

  r = a ** b

  pgcd = r.pgcd_numerateurs()
  ppcm = r.ppcm_denominateurs()

  if visible:
    print(r)

    print(pgcd)
    print(ppcm)

  ok = (pgcd == 1) and (ppcm == 19683)
  return ok
```

```python
def test_unitaire_29(visible =False):
  print("*** polynome: test_unitaire_29 ***")

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

  r = a ** 3
  if visible: print(r.joli())

  ok = (r.valuation() == ra.rationnel(1))
  return ok



def test_unitaire_30(visible =False):
  print("*** polynome: test_unitaire_30 ***")

  ok = True
  return ok



def test_unitaire_31(visible =False):
  print("*** polynome: test_unitaire_31 ***")

  ok = True
  return ok



def test_unitaire_32(visible =False):
  print("*** polynome: test_unitaire_32 ***")

  ok = True
  return ok



def test_unitaire_33(visible =False):
  print("*** polynome: test_unitaire_33 ***")

  ok = True
  return ok



def test_unitaire_34(visible =False):
  print("*** polynome: test_unitaire_34 ***")

  ok = True
  return ok



def test_unitaire_35(visible =False):
  print("*** polynome: test_unitaire_35 ***")

  ok = True
  return ok



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

```
  ok = True
  return ok



def tests_unitaires():
  return (
    test_unitaire_0() and \
    test_unitaire_1() and \
    test_unitaire_2() and \
    test_unitaire_3() and \
    test_unitaire_4() and \
    test_unitaire_5() and \
    test_unitaire_6() and \
    test_unitaire_7() and \
    test_unitaire_8() and \
    test_unitaire_9() and \

    test_unitaire_10() and \
    test_unitaire_11() and \
    test_unitaire_12() 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() and \

    test_unitaire_20() and \
    test_unitaire_21() and \
    test_unitaire_22() and \
    test_unitaire_23() and \
    test_unitaire_24() and \
    test_unitaire_25() and \
    test_unitaire_26() and \
    test_unitaire_27() and \
    test_unitaire_28() and \
    test_unitaire_29(True) and \

    test_unitaire_30() and \
    test_unitaire_31() and \
    test_unitaire_32() and \
    test_unitaire_33() and \
    test_unitaire_34() and \
    test_unitaire_35
  )



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