Initiation à l'algorithmique

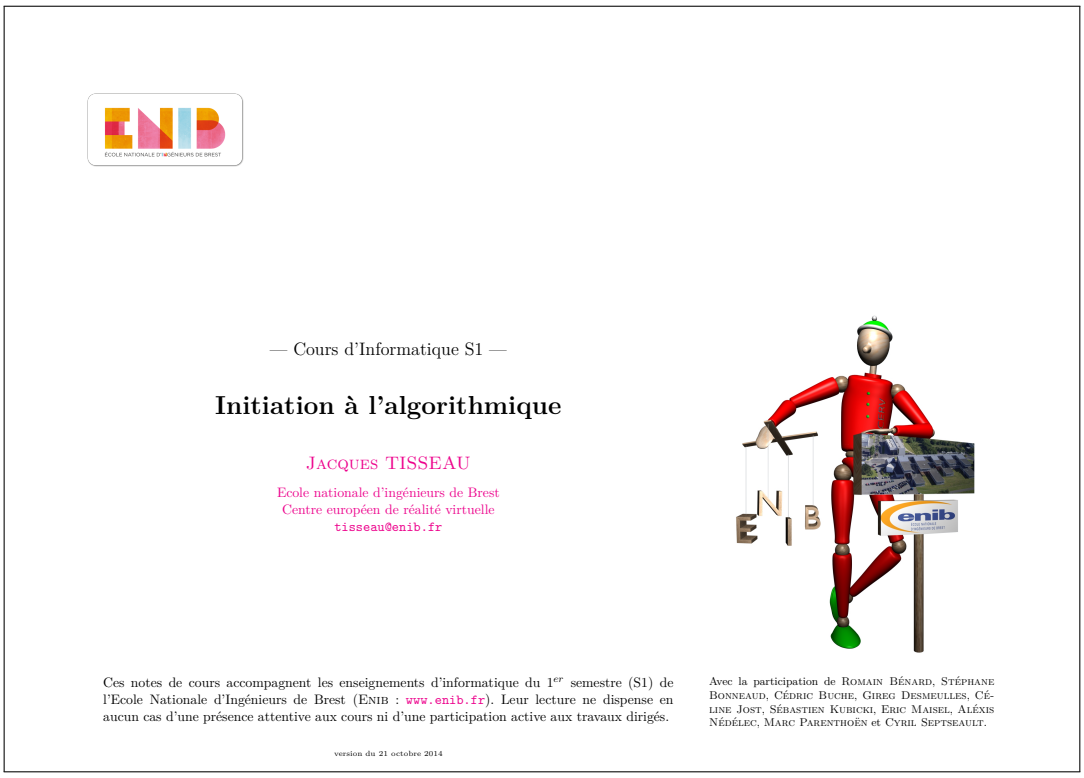
Solutions des questionnements de cours

Jacques TISSEAU

Ecole nationale d'ingénieurs de Brest
Centre européen de réalité virtuelle
tisseau@enib.fr

Avec la participation de Romain Bénard, Stéphane Bonneaud, Cédric Buche, Gireg Desmeulles, Céline Jost, Sébastien Kubicki, Eric Maisel, Aléxis Nédélec, Marc Parenthoën et Cyril Septseault.

Ce document regroupe les solutions, testées avec Python 3.2.2, des questionnements du cours d'Informatique d'informatique du 1^{er} semestre (S1) de l'Ecole Nationale d'Ingénieurs de Brest (ENIB : www.enib.fr). Ils complètent les notes de cours « Initiation à l'algorithmique ».



— Cours d'Informatique S1 —

Initiation à l'algorithmique

Jacques TISSEAU

Ecole nationale d'ingénieurs de Brest
Centre européen de réalité virtuelle
tisseau@enib.fr

Ces notes de cours accompagnent les enseignements d'informatique du 1^{er} semestre (S1) de l'Ecole Nationale d'Ingénieurs de Brest (Enib : www.enib.fr). Leur lecture ne dispense en aucun cas d'une présence attentive aux cours ni d'une participation active aux travaux dirigés.

version du 21 octobre 2014

Avec la participation de Romain Bénard, Stéphane Bonneaud, Cédric Buche, Gireg Desmeulles, Céline Jost, Sébastien Kubicki, Eric Maisel, Aléxis Nédélec, Marc Parenthoën et Cyril Septseault.

Tisseau J.,*Initiation à l'algorithmique*, ENIB, cours d'Informatique S1, Brest, 2009-2014.

Table des matières

1 Introduction

Ce document regroupe uniquement les codes sources PYTHON correspondant aux questionnements [3] du cours d'initiation à l'algorithmique [1] de l'ENIB et non les méthodes et vérifications associées telles que le préconise la démarche MVR [4].

Cette démarche, dite MVR pour Méthode-Vérification-Résultat, est structurée en 3 étapes :

1. on commence par expliciter la méthode générique qui permet de résoudre des problèmes équivalents à celui qui est posé (étape M comme Méthode) ;
2. on explicite ensuite une technique alternative ou complémentaire connue qui permettra de vérifier le résultat obtenu en appliquant la méthode générique précédente (étape V comme Vérification).
3. enfin, on applique la méthode générique (M) et la technique de vérification (V) au cas particulier de l'énoncé pour obtenir le résultat attendu par l'exercice (étape R comme Résultat) ;

La démarche complète est détaillée en cours, au cas par cas, lors de la correction de ces questionnements.

2 Qu'est-ce que l'algorithmique ?

```
# -*- coding: utf-8 -*-

ga  = 0
bu  = 1
zo  = 2
meu = 3

#-------------------------------------------------------------------------
def corrige(a,b):
    assert type(a) is list
    assert type(b) is list

    ca = decodage(a,4)
    cb = decodage(b,4)
    q = codage(ca//cb,4)
    r = codage(ca%cb,4)

    printShadok(a)
    print('÷ ',end=' ')
    printShadok(b)
    print('\t-> quotient = ',end=' ')
    printShadok(q)
    print(', reste = ',end=' ')
    printShadok(r)
    print()

    return q,r

#-------------------------------------------------------------------------
def decodage(code,b) :
    assert type(code) is list
    n = 0
    for i in range(len(code)) :
        n = n + code[len(code)-1-i]*b**i
    return n

#-------------------------------------------------------------------------
def codage(n,b) :
    assert type(n) is int and n >= 0
    assert type(b) is int and b > 1

    if n == 0 : code = [0]
    else :
        code = []
        while n > 0 :
            r = n%b
            n = n//b
            code.insert(0,r)

    return code

#-------------------------------------------------------------------------
def printShadok(code) :
    assert type(code) is list

    for e in code:
```

```
57          if e == 0   : ce = 'ga'
58          elif e == 1 : ce = 'bu'
59          elif e == 2 : ce = 'zo'
60          elif e == 3 : ce = 'meu'
61          else        : ce = 'problème'
62          print(ce,end=' ')
63      return
64
65  #-----------------------------------------------------------------------
66  # 1.
67  a, b = [meu, zo, bu, meu], [zo, zo, zo]
68  corrige(a,b)
69
70  # 2.
71  a, b = [zo, meu, ga, zo], [bu, meu, zo]
72  corrige(a,b)
73
74  # 3.
75  a, b = [bu, ga, zo, meu], [meu, zo, zo]
76  corrige(a,b)
77
78  # 4.
79  a, b = [meu, zo, ga, bu], [meu, zo, zo]
80  corrige(a,b)
81
82  # 5.
83  a, b = [bu, zo, bu, zo], [bu, ga, zo]
84  corrige(a,b)
85
86  # 6.
87  a, b = [bu, ga, meu, bu], [bu, ga, ga]
88  corrige(a,b)
89
90  # 7.
91  a, b = [zo, bu, zo, meu], [zo, ga, bu]
92  corrige(a,b)
93
94  # 8.
95  a, b = [meu, zo, ga, bu], [zo, bu, ga]
96  corrige(a,b)
97
98  # 9.
99  a, b = [bu, meu, meu, ga], [bu, meu, ga]
100 corrige(a,b)
101
102 # 10.
103 a, b = [zo, bu, meu, meu], [bu, ga, bu]
104 corrige(a,b)
105
106 # 11.
107 a, b = [zo, ga, zo, meu], [zo, ga, zo]
108 corrige(a,b)
109
110 # 12.
111 a, b = [bu, meu, bu, ga], [meu, zo, meu]
112 corrige(a,b)
113
114 # 13.
115 a, b = [zo, meu, bu, zo], [bu, ga, meu]
```

```
corrige(a,b)

# 14.
a, b = [zo, ga, meu, meu], [bu, zo, ga]
corrige(a,b)

# 15.
a, b = [meu, meu, ga, zo], [zo, meu, meu]
corrige(a,b)

# 16.
a, b = [bu, ga, zo, meu], [bu, ga, zo]
corrige(a,b)

# 17.
a, b = [zo, zo, ga, meu], [bu, ga, ga]
corrige(a,b)

# 18.
a, b = [zo, meu, meu, ga], [zo, meu, ga]
corrige(a,b)

# 19.
a, b = [zo, bu, bu, zo], [zo, bu, ga]
corrige(a,b)

# 20.
a, b = [meu, bu, bu, zo], [meu, bu, ga]
corrige(a,b)

# 21.
a, b = [meu, ga, zo, meu], [meu, ga, zo]
corrige(a,b)

# 22.
a, b = [bu, bu, zo, zo], [bu, ga, bu]
corrige(a,b)

# 23.
a, b = [bu, zo, ga, bu], [bu, zo, zo]
corrige(a,b)

# 24.
a, b = [meu, ga, meu, bu], [bu, meu, ga]
corrige(a,b)
```

3 Qu'est-ce que l'affectation ?

```
# -*- coding: utf-8 -*-

annee           = 365*24*60*60
anneeLumiere    = 9.46053e15
atmosphere      = 1.01325e5
baril           = 0.15891
calorie         = 4.184
centimetre      = 1.0e-2
electronVolt    = 1.602189e-19
faraday         = 9.6487e4
foot            = 30.48e-2
franklin        = 3.33564e-10
frigorie        = 4.186e3
gallon          = 3.78541e-3
kilometreHeure  = 1.0e3/(60*60)
litre           = 1.0e-3
inch            = 2.54e-2
mile            = 1.609344e3
minute          = 60
noeud           = 0.514444
parsec          = 3.0857e16
pica            = 4.2175e-3
seconde         = 1
torr            = 133.3224

#-------------------------------------------------------------------------
def conversion(a1,a2) :
    print('a1 , a2 = ',eval(a1),',',eval(a2))
    print('n' + a2.title() + ' = n' + a1.title() + ' * a1/a2')
    print()
    return

#-------------------------------------------------------------------------
conversion('annee','minute')
conversion('anneeLumiere','mile')
conversion('baril','gallon')
conversion('anneeLumiere','parsec')
conversion('litre','gallon')
conversion('parsec','inch')
conversion('inch','pica')
conversion('mile','foot')
conversion('frigorie','calorie')
conversion('centimetre','pica')
conversion('electronVolt','frigorie')
conversion('franklin','faraday')
conversion('baril','litre')
conversion('parsec','foot')
conversion('annee','seconde')
conversion('electronVolt','calorie')
conversion('atmosphere','torr')
conversion('parsec','anneeLumiere')
conversion('foot','inch')
conversion('noeud','kilometreHeure')
conversion('mile','inch')
conversion('anneeLumiere','pica')
conversion('inch','foot')
```

```
57 conversion('litre','baril')
```

4 Comment calcule-t-on avec des opérateurs booléens ?

```
# -*- coding: utf-8 -*-

def table(s0,t0,u0,v0,z0) :

    print('a','b','c','|','s','t','u','v','|','z')
    print('----------------|--')
    for a in [0,1] :
        for b in [0,1] :
            for c in [0,1] :
                s = eval(s0)
                t = eval(t0)
                u = eval(u0)
                v = eval(v0)
                z = eval(z0)
                print(a,b,c,'|',int(s),int(t),int(u),int(v),'|',int(z))
    print()
    return

#-------------------------------------------------------------------------
# 1
print('1.')
s = 'not a or b'
t = 'not (not b or c)'
u = 's or t'
v = 'not c or not a'
z = 'not u or v'
table(s,t,u,v,z)

# 2
print('2.')
s = 'not a or b'
t = 'not b or not c'
u = 's and t'
v = '(not c) != (not a)'
z = 'not u or v'
table(s,t,u,v,z)

# 3
print('3.')
s = 'not (a != (not b))'
t = 'b and c'
u = 'not s or t'
v = '(not c) != (not a)'
z = 'not u or v'
table(s,t,u,v,z)

# 4
print('4.')
s = 'not a or not b'
t = 'b or c'
u = 's and t'
v = 'not (c or not a)'
z = 'not u or v'
table(s,t,u,v,z)

# 5
```

```
print('5.')
s = 'a or not b'
t = 'not b or c'
u = 's and t'
v = 'not c or a'
z = 'not u or v'
table(s,t,u,v,z)

# 6
print('6.')
s = 'not a or b'
t = 'not (not b or not c)'
u = 's or t'
v = '(not c) != (not a)'
z = 'not u or v'
table(s,t,u,v,z)

# 7
print('7.')
s = 'a and b'
t = 'b and c'
u = 's != t'
v = 'c or not a'
z = 'not u or v'
table(s,t,u,v,z)

# 8
print('8.')
s = 'not a or b'
t = 'not b or not c'
u = 's and t'
v = 'c or not a'
z = 'not u or v'
table(s,t,u,v,z)

# 9
print('9.')
s = 'not a or b'
t = 'not (not b or c)'
u = 's and t'
v = 'not c or a'
z = 'u != v'
table(s,t,u,v,z)

# 10
print('10.')
s = 'a and b'
t = 'b and c'
u = 'not s or t'
v = 'c or not a'
z = 'not u or v'
table(s,t,u,v,z)

# 11
print('11.')
s = 'not (a != (not b))'
t = 'b and c'
u = 'not s or t'
v = '(not c) != (not a)'
```

```
116 z = 'not u or v'
117 table(s,t,u,v,z)
118
119 # 12
120 print('12.')
121 s = 'a or b'
122 t = 'not (b and c)'
123 u = 's != (not t)'
124 v = 'c or not a'
125 z = 'not u or v'
126 table(s,t,u,v,z)
127
128 # 13
129 print('13.')
130 s = 'not a or b'
131 t = 'b != c'
132 u = 's or t'
133 v = 'c or not a'
134 z = 'not u or v'
135 table(s,t,u,v,z)
136
137 # 14
138 print('14.')
139 s = 'a or not b'
140 t = 'not b or c'
141 u = 's and t'
142 v = 'not c or a'
143 z = 'u != v'
144 table(s,t,u,v,z)
145
146 # 15
147 print('15.')
148 s = 'not a or b'
149 t = 'not b or c'
150 u = 's or t'
151 v = 'c or not a'
152 z = 'not u or v'
153 table(s,t,u,v,z)
154
155 # 16
156 print('16.')
157 s = 'not a or b'
158 t = 'not b or c'
159 u = 's or t'
160 v = 'not c or not a'
161 z = 'not u or v'
162 table(s,t,u,v,z)
163
164 # 17
165 print('17.')
166 s = 'not a or not b'
167 t = 'b != c'
168 u = 's and t'
169 v = 'not (c or not a)'
170 z = 'not u or v'
171 table(s,t,u,v,z)
172
173 # 18
174 print('18.')
```

```
175 s = 'not a or b'
176 t = 'not (not b or c)'
177 u = 's or t'
178 v = 'not c or a'
179 z = 'u != v'
180 table(s,t,u,v,z)
181
182 # 19
183 print('19.')
184 s = 'a != b'
185 t = 'b or c'
186 u = 'not s or t'
187 v = '(not c) != (not a)'
188 z = 'not u or v'
189 table(s,t,u,v,z)
190
191 # 20
192 print('20.')
193 s = 'not a or not b'
194 t = 'not (not b or c)'
195 u = 's and t'
196 v = 'c or a'
197 z = 'not u or v'
198 table(s,t,u,v,z)
199
200 # 21
201 print('21.')
202 s = 'a or b'
203 t = 'not (b and c)'
204 u = 's or t'
205 v = 'c or not a'
206 z = 'not u or v'
207 table(s,t,u,v,z)
208
209 # 22
210 print('22.')
211 s = 'not a or not b'
212 t = 'not (not b or c)'
213 u = 's and t'
214 v = 'c and a'
215 z = 'not u or v'
216 table(s,t,u,v,z)
217
218 # 23
219 print('23.')
220 s = 'a or b'
221 t = 'b or c'
222 u = 'not s or t'
223 v = 'c or not a'
224 z = 'not u or v'
225 table(s,t,u,v,z)
226
227 # 24
228 print('24.')
229 s = 'not a or b'
230 t = 'not (not b or not c)'
231 u = 's and t'
232 v = '(not c) != (not a)'
233 z = 'not u or v'
```

```
234 table(s,t,u,v,z)
```

5 Comment coder un nombre ?

```
# -*- coding: utf-8 -*-

def corrige(x):
    statut, code = ieee754(x)
    print('x = ',x,'\t -> |',code[0],'|',sep='',end='')
    printListe(code[1:9])
    print('|',sep='',end='')
    printListe(code[9:64])
    print('|\n')
    return

#-------------------------------------------------------------------------
def printListe(t) :
    assert type(t) is list
    for e in t :
        print(e,sep='',end='')
    return

# voir TD 3.22
#-------------------------------------------------------------------------
def decodeIeee754(code):
    """
    x = decodeIeee754(code)
    valeur décimale du réel correspondant au code IEEE 754

    >>> decodeIeee754(ieee754(-31.7,2)[1])
    -31.7
    >>> decodeIeee754(ieee754(-31.7e-5,2)[1])
    -0.000317
    >>> decodeIeee754(ieee754(-31.7e5,2)[1])
    -3170000.0
    """
    assert type(code) is list
    assert len(code) == 32 or len(code) == 64

    ke, km, kieee, biais = precision754(len(code)//32)

    signe = (-1)**code[0]

    cexposant = code[1:ke+1]
    cmantisse = code[ke+1:km+ke+1]

    exposant = decodage(cexposant,2) - biais

    mantisse = 1
    for i in range(len(cmantisse)) :
        mantisse = mantisse + cmantisse[i]*2**(-i-1)

    x = signe*mantisse*2**exposant

    return x

#-------------------------------------------------------------------------
def ieee754(x,precision=1) :
    """
    (statut, code) = ieee754(x,precision)
```

```
57      codage selon la norme IEEE 754 du réel x en simple précision
58      (precision == 1) ou en double précision (precision == 2).
59      Si statut == 'normal', code est le code IEEE754 recherché
60      sinon statut = 'underflow' ou 'overflow' et code = [0]
61
62      >>> ieee754(0.0,1)
63      ('normal', [0,\
64   0, 0, 0, 0, 0, 0, 0, 0,\
65   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
66      >>> ieee754(-0.0625,1)
67      ('normal', [1,\
68   0, 1, 1, 1, 1, 0, 1, 1,\
69   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
70      >>> ieee754(-0.0625e-250,1)
71      ('underflow', [1,\
72   0, 0, 0, 0, 0, 0, 0, 0,\
73   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
74      >>> ieee754(-0.0625e250,1)
75      ('overflow', [1,\
76   1, 1, 1, 1, 1, 1, 1, 1,\
77   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
78      >>> ieee754(0.0625,2)
79      ('normal', [0,\
80   0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,\
81   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\
82   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
83      >>> ieee754(-0.0625e-250,2)
84      ('normal', [1,\
85   0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0,\
86   0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1,\
87   0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0])
88       >>> ieee754(173.2679,1)
89       ('normal', [0,\
90   1, 0, 0, 0, 0, 1, 1, 0,\
91   0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1])
92       >>> ieee754(173.2679,2)
93       ('normal', [0,\
94   1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,\
95   0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,\
96   1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0])
97      """
98      assert type(x) is float
99      assert precision in [1,2]
100
101     ke, km, kieee, biais = precision754(precision)
102
103     statut = 'normal'
104     code = [0 for i in range(kieee)]
105     if x != 0.0 :
106         statut, mantisse, exposant = mantisse_exposant(x,precision)
107         code[0]            = signe(x)
108         code[1:ke+2]       = exposant
109         code[ke+1:km+ke+1] = mantisse
110
111     return statut, code
112
113 #-------------------------------------------------------------------------
114 def precision754(precision) :
115     assert precision in [1,2]
```

```
116
117     if precision == 1 :
118         ke, km, kieee, bias = 8, 23, 32, 127
119     else :
120         ke, km, kieee, bias = 11, 52, 64, 1023
121
122     return ke, km, kieee, bias
123
124 #-------------------------------------------------------------------------
125 def mantisse_exposant(x,precision=1) :
126     assert type(x) is float
127     assert precision in [1,2]
128
129     ke, km, kieee, bias = precision754(precision)
130     cbias = codage(bias,2,ke,True)[1]
131
132     pe, pf = partieEntiere(x), partieFractionnaire(x)
133
134     expo, debut = 0, False
135     statut, mantisse = codage(pe,2,km+1)
136
137     if statut == 'overflow' :
138         exposant = codage(2**ke-1,2,ke,True)[1]
139         mantisse = codage(0,2,km,True)[1]
140     else :
141         if mantisse == [0] :
142             debut = True
143             mantisse = []
144
145         expo = len(mantisse)-1
146         i = len(mantisse)
147         while (pf != 0) and (i < km+1):
148             pf = pf * 2
149             pe = int(pf)
150             pf = pf - pe
151             if (pe == 0) and debut == True :
152                 expo = expo - 1
153             else:
154                 debut = False
155                 mantisse.append(pe)
156                 i = i + 1
157
158         if len(mantisse) > 0 and mantisse[0] == 1 :
159             del mantisse[0]
160
161         for i in range(len(mantisse),km):
162             mantisse.append(0)
163
164         if   expo >  bias  :
165             statut = 'overflow'
166             expo = 2**ke-1
167             mantisse = codage(0,2,km,True)[1]
168         elif expo <= -bias :
169             statut = 'underflow'
170             expo = -bias
171             mantisse = codage(0,2,km,True)[1]
172         else               :
173             statut = 'normal'
174
```

```
175         expo = expo + bias
176         exposant = codage(expo,2,ke,True)[1]
177 
178     return statut, mantisse, exposant
179 
180 #-------------------------------------------------------------------------
181 def signe(x) :
182     """
183     s = signe(x)
184     signe du réel x, 0 si x > 0, 1 si x < 0
185     >>> signe(5.7)
186     0
187     >>> signe(-5.7)
188     1
189     """
190     assert type(x) is float
191 
192     return int(x < 0)
193 
194 #-------------------------------------------------------------------------
195 def partieEntiere(x) :
196     """
197     pe = partieEntiere(x)
198     partie entière du réel x
199 
200     >>> partieEntiere(0.67)
201     0
202     >>> partieEntiere(-0.67)
203     0
204     >>> partieEntiere(34.8)
205     34
206     """
207     assert type(x) is float
208 
209     return int(abs(x))
210 
211 #-------------------------------------------------------------------------
212 def partieFractionnaire(x) :
213     """
214     pf = partieFractionnaire(x)
215     partie fractionnaire du réel x
216 
217     >>> partieFractionnaire(8.5)
218     0.5
219     >>> partieFractionnaire(-8.5)
220     0.5
221     >>> partieFractionnaire(0.5)
222     0.5
223     """
224     assert type(x) is float
225 
226     return abs(x) - partieEntiere(x)
227 
228 #-------------------------------------------------------------------------
229 def codage(n,b,k,remplir=False) :
230     """
231     statut, code = codage(n,b,k,remplir)
232     code en base b sur k bits maximum l'entier décimal n
233     statut = 'normal' si 0 <= n < b**k, 'overflow' sinon
```

```
234     si remplir == True, on remplit à gauche de 0 pour avoir len(code) = k
235
236     >>> codage(0,2,8,False)
237     ('normal', [0])
238     >>> codage(0,2,8,True)
239     ('normal', [0, 0, 0, 0, 0, 0, 0, 0])
240     >>> codage(256,2,8,False)
241     ('overflow', [0])
242     >>> codage(65,2,8,False)
243     ('normal', [1, 0, 0, 0, 0, 0, 1])
244     >>> codage(65,2,8,True)
245     ('normal', [0, 1, 0, 0, 0, 0, 0, 1])
246     >>> codage(65,5,4,True)
247     ('normal', [0, 2, 3, 0])
248     >>> codage(79,16,4,True)
249     ('normal', [0, 0, 4, 15])
250     """
251     assert type(n) is int and n >= 0
252     assert type(b) is int and b > 1
253     assert type(k) is int and k > 0
254
255     if n == 0 :
256         statut = 'normal'
257         code = [0]
258     elif n >= b**k :
259         statut = 'overflow'
260         code = [0]
261     else :
262         statut = 'normal'
263         code = []
264         i = 0
265         while (n != 0) and (i < k) :
266                 code.insert(0,n%b)
267                 n = n//b
268                 i = i + 1
269
270     if remplir == True :
271         diffLen = k - len(code)
272         for i in range(diffLen):
273             code.insert(0,0)
274
275     return statut, code
276
277 #-------------------------------------------------------------------------
278 def decodage(code,b=2) :
279     """
280     n = decodage(code,b)
281     valeur décimale de l'entier correspondant au code en base b
282
283     >>> decodage(codage(0,2,8,False)[1],2)
284     0
285     >>> decodage(codage(65,2,8,True)[1],2)
286     65
287     >>> decodage(codage(79,16,4,True)[1],16)
288     79
289     """
290     assert type(code) is list
291     assert type(b) is int and b > 1
292
```

```
293     x = 0
294     for i in range(len(code)) :
295         c = code[len(code)-1-i]
296         x = x + c*b**i
297 
298     return x
299 
300 #-------------------------------------------------------------------------
301 if __name__ == "__main__":
302     import doctest
303     doctest.testmod()
304 
305 #-------------------------------------------------------------------------
306 corrige(-37.03125)
307 
308 corrige(49.1875)
309 
310 corrige(-53.0625)
311 
312 corrige(65.25)
313 
314 corrige(-77.375)
315 
316 corrige(89.75)
317 
318 corrige(-99.625)
319 
320 corrige(7.5)
321 
322 corrige(43.1875)
323 
324 corrige(-13.0625)
325 
326 corrige(71.25)
327 
328 corrige(-54.375)
329 
330 corrige(-19.125)
331 
332 corrige(29.3125)
333 
334 corrige(-37.5625)
335 
336 corrige(45.875)
337 
338 corrige(27.75)
339 
340 corrige(-33.625)
341 
342 corrige(69.5)
343 
344 corrige(-83.125)
345 
346 corrige(99.3125)
347 
348 corrige(-87.5625)
349 
350 corrige(75.875)
351 
```

```
352 corrige(-61.25)
```

6 Qu'est-ce qu'un test ?

```
# -*- coding: utf-8 -*-

x = 0

# 1.
if   x < -4 : y = 0
elif x < -1 : y = -x/3 - 4/3
elif x < 1  : y =  x
elif x < 4  : y =  x/3 + 2/3
else        : y = 2

# 2.
if   x < -4 : y = 0
elif x < -1 : y = x/3 + 4/3
elif x < 1  : y =  -x
elif x < 4  : y =  -x/3 - 2/3
else        : y = -2

# 3.
if   x < -4 : y = -1
elif x < -2 : y = 2
elif x < 0  : y = -3*x/2 - 1
elif x < 1  : y = -1
elif x < 3  : y = 1
else        : y = -1

# 4.
if   x < -4 : y = 1
elif x < -2 : y = -2
elif x < 0  : y = 3*x/2 + 1
elif x < 1  : y = 1
elif x < 3  : y = -1
else        : y = 1

# 5.
if   x < -5 : y = 1
elif x < -3 : y = -x - 4
elif x < -2 : y = 2*x + 5
elif x < -1 : y = -2*x - 3
elif x <  2 : y = x
elif x <  5 : y = -2*x/3 + 10/3
else        : y = 0

# 6.
if   x < -5 : y = 2
elif x < -3 : y = -2*x - 8
elif x < -1 : y = -2
elif x < 1  : y = 2*x
elif x < 4  : y = -4*x/3 + 10/3
elif x < 5  : y = x - 6
else        : y = -1

# 7.
if   x < -3 : y = -1
elif x < -1 : y = 3*x/2 + 7/2
elif x < 2  : y =  -x/3 + 5/3
```

```
57 else        : y = 1
58
59 # 8.
60 if   x < -3 : y = 1
61 elif x < -1 : y = -3*x/2 - 7/2
62 elif x < 2  : y =  x/3 - 5/3
63 else        : y = -1
64
65 # 9.
66 if   x < -5 : y = 0
67 elif x < 0  : y = -2*x/5 - 2
68 elif x < 1  : y = 4*x - 2
69 elif x < 2  : y = -3*x + 5
70 elif x < 3  : y = 3*x - 7
71 elif x < 5  : y = -2*x + 8
72 else        : y = -2
73
74 # 10.
75 if   x < -5 : y = 0
76 elif x < 0  : y = 2*x/5 + 2
77 elif x < 1  : y = -4*x + 2
78 elif x < 2  : y = 3*x - 5
79 elif x < 3  : y = -3*x + 7
80 elif x < 5  : y = 2*x - 8
81 else        : y = 2
82
83 # 11.
84 if   x < -3 : y = -1
85 elif x < -1 : y = 3*x/2 + 7/2
86 elif x < 2  : y =  -x/3 + 5/3
87 else        : y = 1
88
89 # 12.
90 if   x < -4 : y = 1
91 elif x < -2 : y = -x - 3
92 elif x < 2  : y =  x/2
93 elif x < 4  : y =  -x + 3
94 else        : y = -1
95
96 # 13.
97 if   x < -4 : y = -1
98 elif x < -2 : y = x + 3
99 elif x < 2  : y =  -x/2
100 elif x < 4  : y =  x - 3
101 else        : y = 1
102
103 # 14.
104 if   x < -2 : y = 2
105 elif x < -1 : y = -2*x - 2
106 elif x < 1  : y =  x/2 + 1/2
107 elif x < 3  : y =  -x + 2
108 else        : y = -1
109
110 # 15.
111 if   x < -2 : y = -2
112 elif x < -1 : y = 2*x + 2
113 elif x < 1  : y =  -x/2 - 1/2
114 elif x < 3  : y =  x - 2
115 else        : y = 1
```

```
116
117 # 16.
118 if   x < -4 : y = -1
119 elif x < -2 : y = x + 4
120 elif x < 0  : y = 2
121 elif x < 1  : y = -3*x + 2
122 elif x < 3  : y = x - 2
123 else        : y = -1
124
125 # 17.
126 if   x < -5 : y = -1
127 elif x < -3 : y = x + 4
128 elif x < -2 : y = -2*x - 5
129 elif x < -1 : y = 2*x + 3
130 elif x <  2 : y = -x
131 elif x <  5 : y = 2*x/3 - 10/3
132 else        : y = 0
133
134 # 18.
135 if   x < -5 : y = -2
136 elif x < -3 : y = 2*x + 8
137 elif x < -1 : y = 2
138 elif x < 1  : y = -2*x
139 elif x < 4  : y = 4*x/3 - 10/3
140 elif x < 5  : y = -x + 6
141 else        : y = 1
142
143 # 19.
144 if   x < -3 : y = 1
145 elif x < -1 : y = -3*x/2 - 7/2
146 elif x < 2  : y =  x/3 - 5/3
147 else        : y = -1
148
149 # 20.
150 if   x < -4 : y = -2
151 elif x < -3 : y = 4*x + 14
152 elif x < -2 : y = -3*x - 7
153 elif x < 3  : y =  3*x/5 + 1/5
154 else        : y = 2
155
156 # 21.
157 if   x < -4 : y = 2
158 elif x < -3 : y = -4*x - 14
159 elif x < -2 : y = 3*x + 7
160 elif x < 3  : y =  -3*x/5 - 1/5
161 else        : y = -2
162
163 # 22.
164 if   x < -4 : y = 1
165 elif x < -2 : y = -x - 4
166 elif x < 0  : y = -2
167 elif x < 1  : y = 3*x - 2
168 elif x < 3  : y = -x + 2
169 else        : y = 1
170
171 # 23.
172 if   x < -5 : y = 0
173 elif x < -4 : y = x + 5
174 elif x < -3 : y = -2*x - 7
```

```
175 elif x < 1  : y = -3*x/4 - 5/4
176 elif x < 2  : y = -2
177 elif x < 5  : y = x/3 - 8/3
178 else        : y = -1
179
180 # 24.
181 if   x < -5 : y = 0
182 elif x < -4 : y = -x - 5
183 elif x < -3 : y = 2*x + 7
184 elif x < 1  : y = 3*x/4 + 5/4
185 elif x < 2  : y = 2
186 elif x < 5  : y = -x/3 + 8/3
187 else        : y = 1
188
```

7 Comment construit-on une boucle ?

```
# -*- coding: utf-8 -*-

from math import *

#-------------------------------------------------------------------------
def developpement(x,f,g,k0,u0,y0=0,s=1.0e-9) :
    # calcul de f(x)
    k, u = k0, u0
    y = y0 + u
    while fabs(u) > s :
        u = g(u,k,x)
        y = y + u
        k = k + 1
    print(f,x,f(x),y)
    return y

#-------------------------------------------------------------------------
# asin(x)
f = asin
x = 0.25
k, u = 0, x
y = 0
g = lambda u,k,x : u*x*x*(2*k+1)*(2*k+1)/((2*k+2)*(2*k+3))
developpement(x,f,g,k,u,y)

# acos(x)
f = acos
x = 0.25
k, u = 0, -x
y = pi/2
g = lambda u,k,x : u*x*x*(2*k+1)*(2*k+1)/((2*k+2)*(2*k+3))
developpement(x,f,g,k,u,y)

# atan(x)
f = atan
x = 0.25
k, u = 0, x
y = 0
g = lambda u,k,x : -u*x*x*(2*k+1)/(2*k+3)
developpement(x,f,g,k,u,y)

# 1/(1+x)
f = lambda x : 1/(1+x)
x = 0.25
k, u = 0, 1
y = 0
g = lambda u,k,x : -u*x
developpement(x,f,g,k,u,y)

# 1/(1-x)
f = lambda x : 1/(1-x)
x = 0.25
k, u = 0, 1
y = 0
g = lambda u,k,x : u*x
developpement(x,f,g,k,u,y)
```

```
57
58 # 1/(1+x*x)
59 f = lambda x : 1/(1+x*x)
60 x = 0.25
61 k, u = 0, 1
62 y = 0
63 g = lambda u,k,x : -u*x*x
64 developpement(x,f,g,k,u,y)
65
66 # 1/(1-x*x)
67 f = lambda x : 1/(1-x*x)
68 x = 0.25
69 k, u = 0, 1
70 y = 0
71 g = lambda u,k,x : u*x*x
72 developpement(x,f,g,k,u,y)
73
74 # sqrt(1+x)
75 f = lambda x : sqrt(1+x)
76 x = 0.25
77 k, u = 0, 1
78 y = 0
79 g = lambda u,k,x : -u*x*(2*k-1)/(2*(k+1))
80 developpement(x,f,g,k,u,y)
81
82 # 1/sqrt(1+x)
83 f = lambda x : 1/sqrt(1+x)
84 x = 0.25
85 k, u = 0, 1
86 y = 0
87 g = lambda u,k,x : -u*x*(2*k+1)/(2*k+2)
88 developpement(x,f,g,k,u,y)
89
90 # 1/sqrt(1-x*x)
91 f = lambda x : 1/sqrt(1-x*x)
92 x = 0.25
93 k, u = 0, 1
94 y = 0
95 g = lambda u,k,x : u*x*x*(2*k+1)*(2*k+2)/(4*(k+1)*(k+1))
96 developpement(x,f,g,k,u,y)
97
98 # 1/((a - x)**2)
99 a = 6/5
100 f = lambda x : 1/((a - x)**2)
101 x = 0.25
102 k, u = 0, 1/(a*a)
103 y = 0
104 g = lambda u,k,x : u*x*(k+2)/(a*(k+1))
105 developpement(x,f,g,k,u,y)
106
107 # 1/((a - x)**3)
108 a = 6/5
109 f = lambda x : 1/((a - x)**3)
110 x = 0.25
111 k, u = 0, 1/(a*a*a)
112 y = 0
113 g = lambda u,k,x : u*x*(k+3)/(a*(k+1))
114 developpement(x,f,g,k,u,y)
115
```

```
# 1/((a - x)**5)
a = 6/5
f = lambda x : 1/((a - x)**5)
x = 0.25
k, u = 0, 1/(a*a*a*a*a)
y = 0
g = lambda u,k,x : u*x*(k+5)/(a*(k+1))
developpement(x,f,g,k,u,y)

# exp(x)
f = exp
x = 0.25
k, u = 0, 1
y = 0
g = lambda u,k,x : u*x/(k+1)
developpement(x,f,g,k,u,y)

# exp(-x)
f = lambda x : exp(-x)
x = 0.25
k, u = 0, 1
y = 0
g = lambda u,k,x : -u*x/(k+1)
developpement(x,f,g,k,u,y)

# log(1+x)
f = lambda x : log(1+x)
x = 0.25
k, u = 1, x
y = 0
g = lambda u,k,x : -u*x*k/(k+1)
developpement(x,f,g,k,u,y)

# log(1-x)
f = lambda x : log(1-x)
x = 0.25
k, u = 1, -x
y = 0
g = lambda u,k,x : u*x*k/(k+1)
developpement(x,f,g,k,u,y)

# log((1+x)/(1-x))
f = lambda x : log((1+x)/(1-x))
x = 0.25
k, u = 1, 2*x
y = 0
g = lambda u,k,x : u*x*x*(2*k+1)/(2*k+3)
developpement(x,f,g,k,u,y)

# sinh(x)
f = sinh
x = 0.25
k, u = 0, x
y = 0
g = lambda u,k,x : u*x*x/((2*k+2)*(2*k+3))
developpement(x,f,g,k,u,y)

# cosh(x)
f = cosh
```

```
175 x = 0.25
176 k, u = 0, 1
177 y = 0
178 g = lambda u,k,x : u*x*x/((2*k+1)*(2*k+2))
179 developpement(x,f,g,k,u,y)
180
181 # asinh(x)
182 f = asinh
183 x = 0.25
184 k, u = 0, x
185 y = 0
186 g = lambda u,k,x : -u*x*x*(2*k+1)*(2*k+1)/((2*k+2)*(2*k+3))
187 developpement(x,f,g,k,u,y)
188
189 # atanh(x)
190 f = atanh
191 x = 0.25
192 k, u = 0, x
193 y = 0
194 g = lambda u,k,x : u*x*x*(2*k+1)/(2*k+3)
195 developpement(x,f,g,k,u,y)
196
197 # sin(x)
198 f = sin
199 x = 0.25
200 k, u = 0, x
201 y = 0
202 g = lambda u,k,x : -u*x*x/((2*k+2)*(2*k+3))
203 developpement(x,f,g,k,u,y)
204
205 # cos(x)
206 f = cos
207 x = 0.25
208 k, u = 0, 1
209 y = 0
210 g = lambda u,k,x : -u*x*x/((2*k+1)*(2*k+2))
211 developpement(x,f,g,k,u,y)
```

8 Comment imbriquer des boucles ?

```
# -*- coding: utf-8 -*-

from turtle import *
from math import *

#-------------------------------------------------------------------------
def motif(figure,quinconce,x0=0,y0=0,a=0,dx=20,dy=20,n=5,m=4) :
    assert quinconce in [0,1]
    # dessin du motif
    for j in range(m) :
        xl = x0 + quinconce*dx*(j%2)/2
        yl = y0 + j*dy
        # dessin d'une ligne de figures
        nf = n - (j%2)
        if quinconce == 1 : nf = n - (j%2)
        else : nf = n
        for i in range(nf) :
            x, y = xl + i*dx, yl
            # dessin d'une figure
            up()
            goto(x,y)
            setheading(a)
            down()
            # tracé de la figure
            figure(x,y,a,dx,dy)

#-------------------------------------------------------------------------
def f1(x,y,a,dx,dy):
    c, d = 3, dx/2
    for k in range(c) :
        forward(d)
        left(360/c)
    for k in range(c) :
        forward(d)
        right(360/c)
    return

def f2(x,y,a,dx,dy):
    c, d = 4, dx/2
    for k in range(c) :
         forward(d)
         left(360/c)
    up()
    goto(x+d/2,y-d/2)
    setheading(a+45)
    down()
    for k in range(c) :
         forward(d*sqrt(2))
         left(360/c)
    return

def f3(x,y,a,dx,dy):
    c, d = 3, dx/2
    setheading(a+30)
    for k in range(c) :
         forward(d)
```

```
57          left(360/c)
58      up()
59      goto(x+d/sqrt(3),y)
60      setheading(a+90)
61      down()
62      for k in range(c) :
63          forward(d)
64          left(360/c)
65      return
66
67  def f4(x,y,a,dx,dy):
68      c, d = 3, dx/2
69      for k in range(c) :
70          forward(d)
71          left(360/c)
72      setheading(a-60)
73      circle(d/sqrt(3))
74      return
75
76  def f5(x,y,a,dx,dy):
77      c, d = 3, dx/2
78      for n in range(6) :
79          setheading(a+n*60)
80          for k in range(c) :
81              forward(d)
82              left(360/c)
83      return
84
85  def f6(x,y,a,dx,dy):
86      c, d = 3, dx/2
87      setheading(a+30)
88      for k in range(c) :
89          forward(d)
90          left(360/c)
91      up()
92      goto(x+d/sqrt(3),y+d/2)
93      setheading(a-30)
94      down()
95      for k in range(c) :
96          forward(d)
97          left(360/c)
98      return
99
100 def f7(x,y,a,dx,dy):
101     c, d = 3, dx/2
102     for n in range(6) :
103         setheading(a+30+n*60)
104         for k in range(c) :
105             forward(d)
106             left(360/c)
107     return
108
109 def f8(x,y,a,dx,dy):
110     c, d = 3, dx/2
111     for k in range(c) :
112         forward(d)
113         left(360/c)
114     up()
115     goto(x+d/2,y+d/sqrt(3))
```

```
116     setheading(a+60)
117     down()
118     for k in range(c) :
119          forward(d)
120          left(360/c)
121     return
122 
123 def f9(x,y,a,dx,dy):
124     c, d = 3, dx/2
125     setheading(a+30)
126     for k in range(c) :
127         forward(d)
128         left(360/c)
129     setheading(a-90)
130     for k in range(c) :
131         forward(d)
132         left(360/c)
133     return
134 
135 def f10(x,y,a,dx,dy):
136     c, d = 4, dx/2
137     for k in range(c) :
138          forward(d)
139          left(360/c)
140     up()
141     goto(x+d/2,y)
142     setheading(a+45)
143     down()
144     for k in range(c) :
145          forward(d/sqrt(2))
146          left(360/c)
147     return
148 
149 def f11(x,y,a,dx,dy):
150     c, d = 3, dx/2
151     for k in range(c) :
152          forward(d)
153          left(360/c)
154     up()
155     goto(x,y+d/sqrt(3))
156     down()
157     for k in range(c) :
158          forward(d)
159          right(360/c)
160     return
161 
162 def f12(x,y,a,dx,dy):
163     c, d = 3, dx/2
164     for k in range(c) :
165         forward(d)
166         right(360/c)
167     setheading(a-120)
168     circle(d/sqrt(3))
169     return
170 
171 #-----------------------------------------------------------------------
172 for i in range(1,13) :
173     motif(eval('f'+str(i)),0,-650+(i-1)*120,-50)
174     motif(eval('f'+str(i)),1,-650+(i-1)*120,50)
```

9 Comment spécifier une fonction ?

```
# -*- coding: utf-8 -*-

ga  = 0
bu  = 1
zo  = 2
meu = 3

#-------------------------------------------------------------------------
def corrige(a,b):
    assert type(a) is list
    assert type(b) is list

    ca = decodage(a,4)
    cb = decodage(b,4)
    q = codage(ca//cb,4)
    r = codage(ca%cb,4)

    printShadok(a)
    print('÷ ',end=' ')
    printShadok(b)
    print('\t-> quotient = ',end=' ')
    printShadok(q)
    print(', reste = ',end=' ')
    printShadok(r)
    print()

    return q,r

#-------------------------------------------------------------------------
def decodage(code,b) :
    assert type(code) is list
    n = 0
    for i in range(len(code)) :
        n = n + code[len(code)-1-i]*b**i
    return n

#-------------------------------------------------------------------------
def codage(n,b) :
    assert type(n) is int and n >= 0
    assert type(b) is int and b > 1

    if n == 0 : code = [0]
    else :
        code = []
        while n > 0 :
            r = n%b
            n = n//b
            code.insert(0,r)

    return code

#-------------------------------------------------------------------------
def printShadok(code) :
    assert type(code) is list

    for e in code:
```

```
57          if e == 0   : ce = 'ga'
58          elif e == 1 : ce = 'bu'
59          elif e == 2 : ce = 'zo'
60          elif e == 3 : ce = 'meu'
61          else        : ce = 'problème'
62          print(ce,end=' ')
63      return
64
65  #---------------------------------------------------------------------
66  # 1.
67  a, b = [meu, zo, bu, meu], [zo, zo, zo]
68  corrige(a,b)
69
70  # 2.
71  a, b = [zo, meu, ga, zo], [bu, meu, zo]
72  corrige(a,b)
73
74  # 3.
75  a, b = [bu, ga, zo, meu], [meu, zo, zo]
76  corrige(a,b)
77
78  # 4.
79  a, b = [meu, zo, ga, bu], [meu, zo, zo]
80  corrige(a,b)
81
82  # 5.
83  a, b = [bu, zo, bu, zo], [bu, ga, zo]
84  corrige(a,b)
85
86  # 6.
87  a, b = [bu, ga, meu, bu], [bu, ga, ga]
88  corrige(a,b)
89
90  # 7.
91  a, b = [zo, bu, zo, meu], [zo, ga, bu]
92  corrige(a,b)
93
94  # 8.
95  a, b = [meu, zo, ga, bu], [zo, bu, ga]
96  corrige(a,b)
97
98  # 9.
99  a, b = [bu, meu, meu, ga], [bu, meu, ga]
100 corrige(a,b)
101
102 # 10.
103 a, b = [zo, bu, meu, meu], [bu, ga, bu]
104 corrige(a,b)
105
106 # 11.
107 a, b = [zo, ga, zo, meu], [zo, ga, zo]
108 corrige(a,b)
109
110 # 12.
111 a, b = [bu, meu, bu, ga], [meu, zo, meu]
112 corrige(a,b)
113
114 # 13.
115 a, b = [zo, meu, bu, zo], [bu, ga, meu]
```

```
116 corrige(a,b)
117
118 # 14.
119 a, b = [zo, ga, meu, meu], [bu, zo, ga]
120 corrige(a,b)
121
122 # 15.
123 a, b = [meu, meu, ga, zo], [zo, meu, meu]
124 corrige(a,b)
125
126 # 16.
127 a, b = [bu, ga, zo, meu], [bu, ga, zo]
128 corrige(a,b)
129
130 # 17.
131 a, b = [zo, zo, ga, meu], [bu, ga, ga]
132 corrige(a,b)
133
134 # 18.
135 a, b = [zo, meu, meu, ga], [zo, meu, ga]
136 corrige(a,b)
137
138 # 19.
139 a, b = [zo, bu, bu, zo], [zo, bu, ga]
140 corrige(a,b)
141
142 # 20.
143 a, b = [meu, bu, bu, zo], [meu, bu, ga]
144 corrige(a,b)
145
146 # 21.
147 a, b = [meu, ga, zo, meu], [meu, ga, zo]
148 corrige(a,b)
149
150 # 22.
151 a, b = [bu, bu, zo, zo], [bu, ga, bu]
152 corrige(a,b)
153
154 # 23.
155 a, b = [bu, zo, ga, bu], [bu, zo, zo]
156 corrige(a,b)
157
158 # 24.
159 a, b = [meu, ga, meu, bu], [bu, meu, ga]
160 corrige(a,b)
```

10 Qu'est-ce que la récursivité ?

```
# -*- coding: utf-8 -*-

def infix (t) :
    if t != [] :
        infix (t[1])
        print (t[0], end=' ')
        infix (t[2])
    else :
        print (0, end=' ')

    return

#---------------------------------------------------------------------
def postfix (t) :
    if t != [] :
        postfix (t[1])
        postfix (t[2])
        print (t[0], end=' ')
    else :
        print (0, end=' ')

    return

#---------------------------------------------------------------------
# 1.
postfix([1, [], [2, [], [3, [5, [], []], [4, [], []]]]])
print()

# 2.
postfix([1, [3, [5, [], []], []], [2, [], [4, [], []]]])
print()

# 3.
infix([6, [4, [2, [], []], []], [3, [], [1, [], []]]])
print()

# 4.
postfix([2, [4, [], []], [1, [], [6, [], [3, [], []]]]])
print()

# 5.
postfix([1, [2, [4, [], []], [3, [5, [], []], []]], []])
print()

# 6.
postfix([1, [3, [5, [], []], [4, [], []]], [2, [], []]])
print()

# 7.
postfix([2, [4, [], [6, [], []]], [1, [3, [], []], []]])
print()

# 8.
infix([1, [2, [4, [], []], [3, [5, [], []], []]], []])
print()

```

```
# 9.
infix([2, [4, [], [6, [], []]], [1, [3, [], []], []]])
print()

# 10.
infix([5, [3, [1, [], []], []], [4, [], [2, [], []]]])
print()

# 11.
infix([1, [], [2, [], [3, [5, [], []], [4, [], []]]]])
print()

# 12.
postfix([2, [4, [], []], [1, [], [6, [], [3, [], []]]])
print()

# 13.
postfix([1, [3, [], [5, [], []]], [2, [4, [], []], []])
print()

# 14.
infix([2, [4, [], []], [1, [], [6, [], [3, [], []]]])
print()

# 15.
infix([1, [3, [], [5, [], []]], [2, [4, [], []], []])
print()

# 16.
postfix([2, [4, [6, [], []], []], [1, [], [3, [], []]])
print()

# 17.
infix([2, [4, [], []], [1, [], [6, [], [3, [], []]]])
print()

# 18.
postfix([5, [3, [1, [], []], []], [4, [], [2, [], []]])
print()

# 19.
infix([1, [3, [5, [], []], []], [2, [], [4, [], []]])
print()

# 20.
infix([2, [], [1, [4, [], []], [6, [], [3, [], []]]])
print()

# 21.
infix([1, [3, [5, [], []], [4, [], []]], [2, [], []])
print()

# 22.
postfix([6, [4, [2, [], []], []], [3, [], [1, [], []]])
print()

# 23.
postfix([2, [], [1, [4, [], []], [6, [], [3, [], []]]])
print()
```

```
116
117 # 24.
118 infix([2, [4, [6, [], []], []], [1, [], [3, [], []]]])
119 print()
```

11 Comment trier une séquence ?

```
# -*- coding: utf-8 -*-

item1 = ('dupont', 23, 'brest', '06789656')
item2 = ('abgral', 61, 'lille', '06231298')
item3 = ('dupont', 23, 'brest', '02989656')
item4 = ('abgral', 67, 'brest', '06556438')
item5 = ('martin', 38, 'paris', '01674523')
item6 = ('abgral', 67, 'lille', '06231298')

annuaire    = [item1, item2, item3, item4, item5, item6]
croissant   = lambda x,y : x < y
decroissant = lambda x,y : x > y

#-------------------------------------------------------------------------
def triAnnuaire(annuaire,cles,ordre) :
    assert type(cles) is list and len(cles) == 4
    assert ordre == croissant or ordre == decroissant

    cmp = lambda x,y : cmpItems(x,y,cles,ordre)
    triInsertion(annuaire,cmp)
    return


#-------------------------------------------------------------------------
def triInsertion (t,cmp =( lambda x,y : x < y)) :
    assert type(t) is list
    for i in range (1, len(t)):
        x = t[i]
        k = i
        for j in range(i -1, -1 , -1):
            if not cmp(t[j],x) :
                k = k-1
                t[j+1] = t[j]
        t[k] = x
    return

#-------------------------------------------------------------------------
def cmpItems (i1 ,i2 ,cles =[0 ,1 ,2 ,3] , cmp =( lambda x,y : x < y)) :
    assert type(i1) is tuple and len(i1) == 4
    assert type(i2) is tuple and len(i2) == 4
    assert type(cles) is list and (0 < len(cles) <= 4)
    for k in cles:
        if cmp(i1[k],i2[k]) : return True
        elif i1[k] == i2[k] : pass
        else                : return False
    return False

#-------------------------------------------------------------------------
def printListe(t) :
    assert type(t) is list
    for e in t :
        print(e)
    return

def corrige(cles,ordre) :
    annuaire    = [item1, item2, item3, item4, item5, item6]
```

```
57      triAnnuaire(annuaire,cles,ordre)
58      print(cles,end=' ')
59      if ordre == croissant : print('croissant')
60      else                  : print('décroissant')
61      printListe(annuaire)
62      print()
63      return
64
65  #-------------------------------------------------------------------------
66  # 1
67  cles, ordre = [3, 0, 1, 2], croissant
68  corrige(cles,ordre)
69
70  # 2.
71  cles, ordre = [3, 0, 2, 1], croissant
72  corrige(cles,ordre)
73
74  # 3.
75  cles, ordre = [3, 1, 0, 2], croissant
76  corrige(cles,ordre)
77
78  # 4.
79  cles, ordre = [3, 1, 2, 0], croissant
80  corrige(cles,ordre)
81
82  # 5.
83  cles, ordre = [3, 2, 0, 1], croissant
84  corrige(cles,ordre)
85
86  # 6.
87  cles, ordre = [3, 2, 1, 0], croissant
88  corrige(cles,ordre)
89
90  # 7.
91  cles, ordre = [1, 0, 2, 3], decroissant
92  corrige(cles,ordre)
93
94  # 8.
95  cles, ordre = [1, 0, 3, 2], decroissant
96  corrige(cles,ordre)
97
98  # 9.
99  cles, ordre = [1, 2, 0, 3], decroissant
100 corrige(cles,ordre)
101
102 # 10.
103 cles, ordre = [1, 2, 3, 0], decroissant
104 corrige(cles,ordre)
105
106 # 11.
107 cles, ordre = [1, 3, 0, 2], decroissant
108 corrige(cles,ordre)
109
110 # 12.
111 cles, ordre = [1, 3, 2, 0], decroissant
112 corrige(cles,ordre)
113
114 # 13.
115 cles, ordre = [2, 0, 1, 3], croissant
```

```
116 corrige(cles,ordre)
117
118 # 14.
119 cles, ordre = [2, 0, 3, 1], croissant
120 corrige(cles,ordre)
121
122 # 15.
123 cles, ordre = [2, 1, 0, 3], croissant
124 corrige(cles,ordre)
125
126 # 16.
127 cles, ordre = [2, 1, 3, 0], croissant
128 corrige(cles,ordre)
129
130 # 17.
131 cles, ordre = [2, 3, 0, 1], croissant
132 corrige(cles,ordre)
133
134 # 18.
135 cles, ordre = [2, 3, 1, 0], croissant
136 corrige(cles,ordre)
137
138 # 19.
139 cles, ordre = [0, 1, 2, 3], decroissant
140 corrige(cles,ordre)
141
142 # 20.
143 cles, ordre = [0, 1, 3, 2], decroissant
144 corrige(cles,ordre)
145
146 # 21.
147 cles, ordre = [0, 2, 1, 3], decroissant
148 corrige(cles,ordre)
149
150 # 22.
151 cles, ordre = [0, 2, 3, 1], decroissant
152 corrige(cles,ordre)
153
154 # 23.
155 cles, ordre = [0, 3, 1, 2], decroissant
156 corrige(cles,ordre)
157
158 # 24.
159 cles, ordre = [0, 3, 2, 1], decroissant
160 corrige(cles,ordre)
161
```

12 Tout en un ?

```
# -*- coding: utf-8 -*-


#---------------------------------------------------------------------------
def typeNombre(x) :
    """
    ok = typeNombre(x)
    True si x est un nombre, False sinon

    >>> typeNombre(3)
    True
    >>> typeNombre(3.14)
    True
    >>> typeNombre(3.14e-5)
    True
    >>> typeNombre('3.14')
    False
    >>> typeNombre([3])
    False
    """

    return type(x) is int or type(x) is float

#---------------------------------------------------------------------------
# piles
def pileVide(pile) :
    """
    ok = pileVide(p)
    True si la pile p est vide, False sinon

    >>> pileVide([])
    True
    >>> pileVide([1,2,3])
    False
    """
    assert type(pile) is list
    return pile == []

def sommet(pile) :
    """
    x = sommet(p)
    sommet de la pile p non vide

    >>> sommet([1,2,3])
    3
    """
    assert not pileVide(pile)
    return pile[len(pile)-1]

def empiler(pile,element) :
    """
    empiler(p,e)
    empile e sur la pile p

    >>> p = [1,2,3]
    >>> empiler(p,7)
```

```
57      >>> p
58      [1, 2, 3, 7]
59      >>> p = []
60      >>> empiler(p,7)
61      >>> p
62      [7]
63      """
64      assert type(pile) is list
65      pile.append(element)
66      return
67
68  def depiler(pile) :
69      """
70      s = depiler(p)
71      dépile le sommet s de la pile p non vide
72
73      >>> p = [1,2,3]
74      >>> depiler(p)
75      3
76      >>> depiler(p)
77      2
78      >>> depiler(p)
79      1
80      """
81      assert not pileVide(pile)
82      s = sommet(pile)
83      del pile[len(pile)-1]
84      return s
85
86  #--------------------------------------------------------------------------
87  # files
88  def fileVide(file) :
89      """
90      ok = fileVide(f)
91      True si la file f est vide, False sinon
92
93      >>> fileVide([])
94      True
95      >>> fileVide([1,2,3])
96      False
97      """
98      assert type(file) is list
99      return file == []
100
101 def tete(file) :
102     """
103     x = tete(f)
104     tete de la file f non vide
105
106     >>> tete([1,2,3])
107     3
108     """
109     assert not fileVide(file)
110     return file[len(file)-1]
111
112 def enfiler(file,element) :
113     """
114     enfiler(f,e)
115     enfile e dans la file f
```

```
116
117     >>> f = [1,2,3]
118     >>> enfiler(f,7)
119     >>> f
120     [7, 1, 2, 3]
121     >>> f = []
122     >>> enfiler(f,7)
123     >>> f
124     [7]
125     """
126     assert type(file) is list
127     file.insert(0,element)
128     return
129
130 def defiler(file) :
131     """
132     t = defiler(f)
133     défile la tête t de la file f non vide
134
135     >>> f = [1,2,3]
136     >>> defiler(f)
137     3
138     >>> defiler(f)
139     2
140     >>> defiler(f)
141     1
142     """
143     assert not fileVide(file)
144     t = tete(file)
145     del file[len(file)-1]
146     return t
147
148 #-------------------------------------------------------------------------
149 # graphes
150 # un graphe est une liste d'arcs
151 # un arc est un triplet (noeud1,noeud2,poids)
152
153 def Arc(arc) :
154     """
155     ok = Arc(a)
156     True si a est un arc, False sinon
157
158     >>> Arc(('a','b',5))
159     True
160     >>> Arc(([1,2],[3,4],'p'))
161     True
162     >>> Arc(('a','b'))
163     False
164     >>> Arc(['a','b',5])
165     False
166     """
167     ok = True
168     if type(arc) is not tuple or len(arc) != 3 :
169         ok = False
170     return ok
171
172 def Graphe(graphe) :
173     """
174     ok = Graphe(g)
```

```
175     True si g est un graphe, False sinon
176
177     >>> g = [('n1','n1',4),('n1','n2',3)]
178     >>> Graphe(g)
179     True
180     >>> Graphe([1,2,3])
181     False
182     >>> Graphe(('n1','n2',3))
183     False
184     """
185     if type(graphe) is not list :
186         return False
187     else :
188         for arc in graphe :
189             if not Arc(arc) :
190                 return False
191     return True
192
193 def adjacents(noeud1,noeud2,graphe) :
194     """
195     ok = adjacents(n1,n2,g)
196     True si n1 et n2 sont 2 noeuds adjacents du graphe g,
197     False sinon
198
199     >>> g = [('n1','n2',4),('n2','n3',3)]
200     >>> adjacents('n1','n2',g)
201     True
202     >>> adjacents('n1','n3',g)
203     False
204     >>> adjacents('n1','n4',g)
205     False
206     >>> adjacents('n2','n1',g)
207     True
208     >>> adjacents('n2','n3',g)
209     True
210     >>> adjacents('n3','n1',g)
211     False
212     >>> adjacents('n3','n2',g)
213     True
214     """
215     assert Graphe(graphe)
216     for (n1,n2,p) in graphe :
217         if (n1 == noeud1 and n2 == noeud2) or \
218            (n2 == noeud1 and n1 == noeud2) :
219             return True
220     return False
221
222 def listeAdjacents(noeud,graphe) :
223     """
224     arcs = listeAdjacents(n,g)
225     liste de tous les noeuds adjacents à un noeund n du graphe g
226
227     >>> g = [('n1','n2',4),('n2','n3',3)]
228     >>> listeAdjacents('n1',g)
229     [('n1', 'n2', 4)]
230     >>> listeAdjacents('n4',g)
231     []
232     >>> listeAdjacents('n2',g)
233     [('n1', 'n2', 4), ('n2', 'n3', 3)]
```

```
234     >>> listeAdjacents('n3',g)
235     [('n2', 'n3', 3)]
236     """
237     assert Graphe(graphe)
238     arcs = []
239     for (noeud1,noeud2,poids) in graphe :
240         if noeud1 == noeud or noeud2 == noeud :
241             arcs.append((noeud1,noeud2,poids))
242     return arcs
243 
244 #-------------------------------------------------------------------------
245 # chemins dans un graphe
246 def Chemin(chemin) :
247     """
248     ok = Chemin(c)
249     True si c est un chemin, False sinon
250 
251     >>> Chemin((7,['n1','n2','n3']))
252     True
253     >>> Chemin((7,[]))
254     False
255     """
256     return type(chemin) is tuple    and \
257            len(chemin) == 2         and \
258            typeNombre(chemin[0])    and \
259            type(chemin[1]) is list  and \
260            len(chemin[1]) > 0
261 
262 def extremite(chemin) :
263     """
264     n = extremite(c)
265     dernier noeud n du chemin c
266 
267     >>> extremite((7,['n1','n2','n3']))
268     'n3'
269     """
270     assert Chemin(chemin)
271     return chemin[1][len(chemin[1])-1]
272 
273 def cheminsSuivants(chemin,graphe) :
274     """
275     cs = cheminSuivants(c,g)
276     liste des chemins possibles à partir du noeud extrémité d'un chemin c
277     d'un graphe g sans repasser par l'un des noeuds du chemin
278 
279     >>> g = [('n1','n2',4),('n2','n3',3),\
280              ('n1','n4',6),('n2','n4',5),\
281              ('n2','n4',6)]
282     >>> cheminsSuivants((0,['n1']),g)
283     [(4, ['n1', 'n2']), (6, ['n1', 'n4'])]
284     >>> cheminsSuivants((6,['n1','n4']),g)
285     [(11, ['n1', 'n4', 'n2']), (12, ['n1', 'n4', 'n2'])]
286     >>> cheminsSuivants((4,['n1','n2']),g)
287     [(7, ['n1', 'n2', 'n3']), (9, ['n1', 'n2', 'n4']), (10, ['n1', 'n2', 'n4'])]
288     >>> cheminsSuivants((7,['n1','n2','n3']),g)
289     []
290     >>> cheminsSuivants((9,['n1','n2','n4']),g)
291     []
292     """
```

```
293     assert Chemin(chemin)
294     assert Graphe(graphe)
295     noeud = extremite(chemin)
296     cout = chemin[0]
297     route = chemin[1]
298     suivants = []
299     for arc in listeAdjacents(noeud,graphe) :
300         if arc[0] == noeud : noeudSuivant = arc[1]
301         else : noeudSuivant = arc[0]
302         if noeudSuivant not in route :
303             routeSuivante = route + [noeudSuivant]
304             coutSuivant = cout + arc[2]
305             suivants.append((coutSuivant,routeSuivante))
306     return suivants
307 
308 #-------------------------------------------------------------------------
309 # recherches de chemins dans un graphe entre noeud1 et noeud2
310 
311 def profondeur(noeud1,noeud2,graphe,n=100) :
312     """
313     cs = profondeur(n1,n2,g,n)
314     liste des n chemins maximum qui mènent du noeud n1 au noeud n2
315     dans le graphe g
316 
317     >>> graphe = [('s','d',4), ('s','a',3), ('d','e',2), ('d','a',5),\
318                   ('a','b',4), ('b','c',4), ('b','e',5), ('e','f',4),\
319                   ('f','g',3)]
320     >>> profondeur('s','g',graphe)
321     [(19, ['s', 'a', 'b', 'e', 'f', 'g']),\
322  (17, ['s', 'a', 'd', 'e', 'f', 'g']),\
323  (25, ['s', 'd', 'a', 'b', 'e', 'f', 'g']),\
324  (13, ['s', 'd', 'e', 'f', 'g'])]
325     >>> profondeur('d','b',graphe)
326     [(9, ['d', 'a', 'b']), (7, ['d', 'e', 'b']), (11, ['d', 's', 'a', 'b'])]
327     """
328     assert Graphe(graphe)
329     assert type(n) is int and n >= 0
330     pile = [(0,[noeud1])]
331     chemins = []
332     while not pileVide(pile) and n > 0 :
333         chemin = depiler(pile)
334         noeud = extremite(chemin)
335         if noeud == noeud2 :
336             chemins.append(chemin)
337             n = n - 1
338         else :
339             pile = pile + cheminsSuivants(chemin,graphe)
340     return chemins
341 
342 def largeur(noeud1,noeud2,graphe,n=100) :
343     """
344     cs = largeur(n1,n2,g,n)
345     liste des n chemins maximum qui mènent du noeud n1 au noeud n2
346     dans le graphe g
347 
348     >>> graphe = [('s','d',4), ('s','a',3), ('d','e',2), ('d','a',5),\
349                   ('a','b',4), ('b','c',4), ('b','e',5), ('e','f',4),\
350                   ('f','g',3)]
351     >>> largeur('s','g',graphe)
```

```
352     [(13, ['s', 'd', 'e', 'f', 'g']),\
353  (19, ['s', 'a', 'b', 'e', 'f', 'g']),\
354  (17, ['s', 'a', 'd', 'e', 'f', 'g']),\
355  (25, ['s', 'd', 'a', 'b', 'e', 'f', 'g'])]
356     >>> largeur('d','b',graphe)
357     [(9, ['d', 'a', 'b']), (7, ['d', 'e', 'b']), (11, ['d', 's', 'a', 'b'])]
358     """
359     assert Graphe(graphe)
360     file = [(0,[noeud1])]
361     chemins = []
362     while not fileVide(file) and n > 0 :
363         chemin = defiler(file)
364         noeud = extremite(chemin)
365         if noeud == noeud2 :
366             chemins.append(chemin)
367             n = n - 1
368         else :
369             file = cheminsSuivants(chemin,graphe) + file
370     return chemins
371 
372 def meilleur(noeud1,noeud2,graphe,n=100) :
373     """
374     cs = meilleur(n1,n2,g,n)
375     liste des n chemins maximum qui mènent du noeud n1 au noeud n2
376     dans le graphe g
377 
378     >>> graphe = [('s','d',4), ('s','a',3), ('d','e',2), ('d','a',5),\
379                   ('a','b',4), ('b','c',4), ('b','e',5), ('e','f',4),\
380                   ('f','g',3)]
381     >>> meilleur('s','g',graphe)
382     [(13, ['s', 'd', 'e', 'f', 'g']),\
383  (17, ['s', 'a', 'd', 'e', 'f', 'g']),\
384  (19, ['s', 'a', 'b', 'e', 'f', 'g']),\
385  (25, ['s', 'd', 'a', 'b', 'e', 'f', 'g'])]
386     >>> meilleur('d','b',graphe)
387     [(7, ['d', 'e', 'b']), (9, ['d', 'a', 'b']), (11, ['d', 's', 'a', 'b'])]
388     >>> meilleur('s','g',graphe,1)
389     [(13, ['s', 'd', 'e', 'f', 'g'])]
390     >>> meilleur('s','g',graphe,2)
391     [(13, ['s', 'd', 'e', 'f', 'g']), (17, ['s', 'a', 'd', 'e', 'f', 'g'])]
392     """
393     assert Graphe(graphe)
394     file = [(0,[noeud1])]
395     chemins = []
396     while not fileVide(file) and n > 0 :
397         chemin = defiler(file)
398         noeud = extremite(chemin)
399         if noeud == noeud2 :
400             chemins.append(chemin)
401             n = n - 1
402         else :
403             file = cheminsSuivants(chemin,graphe) + file
404             file.sort(reverse=True)
405     return chemins
406 
407 
408 #-------------------------------------------------------------------------
409 # problème du réseau routier
410 depart, arrivee = 's', 'g'
```

```
411
412 def reseau(noeud) :
413     """
414     >>> reseau('a')
415     [('s', 'a', 3), ('d', 'a', 5), ('a', 'b', 4)]
416     >>> reseau('s')
417     [('s', 'd', 4), ('s', 'a', 3)]
418     >>> reseau('c')
419     [('b', 'c', 4)]
420     """
421     g = [('s','d',4),\
422          ('s','a',3),\
423          ('d','e',2),\
424          ('d','a',5),\
425          ('a','b',4),\
426          ('b','c',4),\
427          ('b','e',5),\
428          ('e','f',4),\
429          ('f','g',3)]
430     return listeAdjacents(noeud,g)
431
432 #--------------------------------------------------------------------------
433 # problème des vases
434 depart, arrivee = (0,0), (4,0)
435
436 def capacite() :
437     return (8,5)
438
439 def vider(eprouvette, etat) :
440     if eprouvette == 0 :
441         etatFinal = (0,etat[1])
442     else :
443         etatFinal = (etat[0],0)
444     return etatFinal
445
446 def remplir(eprouvette, etat) :
447     if eprouvette == 0 :
448         etatFinal = (capacite()[0],etat[1])
449     else :
450         etatFinal = (etat[0],capacite()[1])
451     return etatFinal
452
453 def transvaser(eprouvette, etat) :
454     if eprouvette == 0 :
455         reste = capacite()[1] - etat[1]
456         if etat[0] <= reste :
457             etatFinal = (0,etat[1]+etat[0])
458         else :
459             etatFinal = (etat[0]-reste,capacite()[1])
460     else :
461         reste = capacite()[0] - etat[0]
462         if etat[1] <= reste :
463             etatFinal = (etat[0]+etat[1],0)
464         else :
465             etatFinal = (capacite()[0],etat[1]-reste)
466     return etatFinal
467
468 def vases(noeud) :
469     """
```

```
    >>> vases((0,0))
    [((0, 0), (0, 5), 1), ((0, 0), (8, 0), 1)]
    >>> vases((8,0))
    [((8, 0), (3, 5), 1), ((8, 0), (8, 5), 1), ((8, 0), (0, 0), 1)]
    >>> vases((0,5))
    [((0, 5), (5, 0), 1), ((0, 5), (8, 5), 1), ((0, 5), (0, 0), 1)]
    """
    suivants = []
    if noeud[1] != 0 and noeud[0] != capacite()[0] :
        noeudSuivant = transvaser(1,noeud)
        suivants.append((noeud,noeudSuivant,1))
    if noeud[0] != 0 and noeud[1] != capacite()[1] :
        noeudSuivant = transvaser(0,noeud)
        suivants.append((noeud,noeudSuivant,1))
    if noeud[1] != capacite()[1] :
        noeudSuivant = remplir(1,noeud)
        suivants.append((noeud,noeudSuivant,1))
    if noeud[0] != capacite()[0] :
        noeudSuivant = remplir(0,noeud)
        suivants.append((noeud,noeudSuivant,1))
    if noeud[1] != 0 :
        noeudSuivant = vider(1,noeud)
        suivants.append((noeud,noeudSuivant,1))
    if noeud[0] != 0 :
        noeudSuivant = vider(0,noeud)
        suivants.append((noeud,noeudSuivant,1))
    return suivants


#-------------------------------------------------------------------------
# jeu du taquin
depart, arrivee = [[3,1],[0,2]], [[1,2],[3,0]]

def carreauVide(jeu) :
    n = len(jeu)
    ok = False
    i = 0
    while i < n and not ok :
        if 0 in jeu[i] :
            j = 0
            while j < n and not ok :
                if jeu[i][j] == 0 : ok = True
                else : j = j + 1
        else : i = i + 1
    return (i,j)

def permuterVide(jeu,pos1,pos2) :
    n = len(jeu)
    jeu1 = []
    for i in range(n) :
        jeu1.append([])
        for j in range(n) :
            jeu1[i].append(jeu[i][j])
    jeu1[pos1[0]][pos1[1]], jeu1[pos2[0]][pos2[1]] = jeu1[pos2[0]][pos2[1]], jeu1[pos1[0]][pos1[1]]
    return jeu1

def taquin(jeu) :
    """
    >>> jeu = [[3,1],[0,2]]
```

```
529     >>> taquin(jeu)
530     [([[3, 1], [0, 2]], [[0, 1], [3, 2]], 1),\
531  ([[3, 1], [0, 2]], [[3, 1], [2, 0]], 1)]
532     >>> jeu = [[2,1,8],[6,0,3],[7,5,4]]
533     >>> taquin(jeu)
534     [([[2, 1, 8], [6, 0, 3], [7, 5, 4]], [[2, 0, 8], [6, 1, 3], [7, 5, 4]], 1),\
535  ([[2, 1, 8], [6, 0, 3], [7, 5, 4]], [[2, 1, 8], [6, 5, 3], [7, 0, 4]], 1),\
536  ([[2, 1, 8], [6, 0, 3], [7, 5, 4]], [[2, 1, 8], [0, 6, 3], [7, 5, 4]], 1),\
537  ([[2, 1, 8], [6, 0, 3], [7, 5, 4]], [[2, 1, 8], [6, 3, 0], [7, 5, 4]], 1)]
538     """
539     n = len(jeu)
540     (i0,j0) = carreauVide(jeu)
541     suivants = []
542 
543     (i1,j1) = (max(0,i0-1),j0)
544     jeu1 = permuterVide(jeu,(i0,j0),(i1,j1))
545     if jeu1 != jeu : suivants.append((jeu,jeu1,1))
546 
547     (i1,j1) = (min(i0+1,n-1),j0)
548     jeu1 = permuterVide(jeu,(i0,j0),(i1,j1))
549     if jeu1 != jeu : suivants.append((jeu,jeu1,1))
550 
551     (i1,j1) = (i0,max(0,j0-1))
552     jeu1 = permuterVide(jeu,(i0,j0),(i1,j1))
553     if jeu1 != jeu : suivants.append((jeu,jeu1,1))
554 
555     (i1,j1) = (i0,min(j0+1,n-1))
556     jeu1 = permuterVide(jeu,(i0,j0),(i1,j1))
557     if jeu1 != jeu : suivants.append((jeu,jeu1,1))
558 
559     return suivants
560 
561 
562 #-------------------------------------------------------------------------
563 def meilleurFct(noeud1,noeud2,f,n=100) :
564     """
565     cs = meilleurf(n1,n2,f,n)
566     liste des n chemins maximum qui mènent du noeud n1 au noeud n2
567 
568     >>> depart, arrivee = 's', 'g'
569     >>> meilleurFct(depart,arrivee,reseau,1)
570     [(13, ['s', 'd', 'e', 'f', 'g'])]
571     >>> meilleurFct('d','b',reseau,1)
572     [(7, ['d', 'e', 'b'])]
573     >>> depart, arrivee = (0,0), (4,0)
574     >>> meilleurFct(depart,arrivee,vases,5)
575     [(12, [(0, 0), (0, 5), (5, 0), (5, 5), (8, 2), (0, 2), (2, 0), (2, 5), (7, 0), (7, 5), (8, 4), (0, 4), (4, 
576  (13, [(0, 0), (8, 0), (3, 5), (3, 0), (0, 3), (8, 3), (6, 5), (6, 0), (1, 5), (1, 0), (0, 1), (8, 1), (4, 5), 
577  (14, [(0, 0), (8, 0), (3, 5), (0, 5), (5, 0), (5, 5), (8, 2), (0, 2), (2, 0), (2, 5), (7, 0), (7, 5), (8, 4), 
578  (14, [(0, 0), (8, 0), (8, 5), (0, 5), (5, 0), (5, 5), (8, 2), (0, 2), (2, 0), (2, 5), (7, 0), (7, 5), (8, 4), 
579  (15, [(0, 0), (0, 5), (5, 0), (8, 0), (3, 5), (3, 0), (0, 3), (8, 3), (6, 5), (6, 0), (1, 5), (1, 0), (0, 1), 
580     >>> depart, arrivee = [[3,1],[0,2]], [[1,2],[3,0]]
581     >>> meilleurFct(depart,arrivee,taquin,10)
582     [(3, [[[3, 1], [0, 2]], [[0, 1], [3, 2]], [[1, 0], [3, 2]], [[1, 2], [3, 0]]]),\
583  (9, [[[3, 1], [0, 2]], [[3, 1], [2, 0]], [[3, 0], [2, 1]], [[0, 3], [2, 1]],\
584  [[2, 3], [0, 1]], [[2, 3], [1, 0]], [[2, 0], [1, 3]], [[0, 2], [1, 3]],\
585  [[1, 2], [0, 3]], [[1, 2], [3, 0]]])]
586     """
587     file = [(0,[noeud1])]
```

```
588     chemins = []
589     while not fileVide(file) and n > 0 :
590         chemin = defiler(file)
591         noeud = extremite(chemin)
592         if noeud == noeud2 :
593             chemins.append(chemin)
594             n = n - 1
595         else :
596             file = cheminsSuivantsFct(chemin,f) + file
597             file.sort(reverse=True)
598     return chemins
599 
600 def cheminsSuivantsFct(chemin,f) :
601     assert Chemin(chemin)
602     noeud = extremite(chemin)
603     cout = chemin[0]
604     route = chemin[1]
605     suivants = []
606     for arc in f(noeud) :
607         if arc[0] == noeud : noeudSuivant = arc[1]
608         else : noeudSuivant = arc[0]
609         if noeudSuivant not in route :
610             routeSuivante = route + [noeudSuivant]
611             coutSuivant = cout + arc[2]
612             suivants.append((coutSuivant,routeSuivante))
613     return suivants
614 
615 #-------------------------------------------------------------------------
616 if __name__ == '__main__' :
617     import doctest
618     doctest.testmod()
```

Références

[1] Tisseau J., *Initiation à l'algorithmique. Cours*, ENIB, 2009-2014

[2] Tisseau J., *Initiation à l'algorithmique. Travaux dirigés*, ENIB, 2009-2014

[3] Tisseau J., *Initiation à l'algorithmique. Questionnments de cours*, ENIB, 2011-2014

[4] Tisseau J., Nédélec A., Parenthoën M., *Initiation à l'algorithmique. La démarche* MVR *: méthode, vérification, résultat*, ENIB, 2011-2014