```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  7 18:47:50 2016

@author: stephane
"""

from time import time

# Exo 1 :


print(5*3)

# Exo 2


def facto(n):
    if n == 0:
        return 1
    return n * facto(n-1)


def expo_basique(x, n):
    if n == 0:
        return 1
    return x * expo_basique(x, n-1)


def expo_rapide(x, n):
    if n == 0:
        return 1
    if n % 2 == 0:
        return expo_basique(x**2, n//2)
    return x * expo_basique(x**2, n//2)

"""
>>> facto(5)
120
>>> expo_basique(42, 13)
1265437718438866624512L
>>> expo_rapide(42, 13)
1265437718438866624512L
>>> 42**13
1265437718438866624512L
"""

# Exo 3


def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

les_temps = []

for n in range(30, 32):  # (30, 36)
    t0 = time()
    poubelle = fibonacci(n)
    t1 = time()
    les_temps.append(t1 - t0)
```



```
rapports = [les_temps[k+1]/les_temps[k] for k in range(len(les_temps) - 1)]

"""
>>> les_temps
[0.6121728420257568, 1.0113229751586914, 1.5932409763336182, 2.634416103363037,
4.124413013458252, 6.769514083862305]
>>> rapports
[1.6520219547997206, 1.5754027303528977, 1.6534950723056228,
1.5655890533743388, 1.6413278839371568]
"""

# Exo 4


def u(n):
    if n == 0:
        return 1.
    return (u(n-1)+2/u(n-1))/2
"""
>>> [u(n) for n in range(10)]
[1.0, 1.5, 1.4166666666666665, 1.4142156862745097, 1.4142135623746899,
1.414213562373095, 1.414213562373095, 1.414213562373095, 1.414213562373095,
1.414213562373095]
"""
les_temps = []

for n in range(15, 18):  # (15, 23)
    t0 = time()
    _ = u(n)
    t1 = time()
    les_temps.append(t1 - t0)

rapports = [les_temps[k+1]/les_temps[k] for k in range(len(les_temps) - 1)]

"""
>>> les_temps
[0.021079063415527344, 0.04273486137390137, 0.0836029052734375,
0.17234086990356445, 0.3387107849121094, 0.6810169219970703,
1.3614799976348877, 2.735727071762085]
>>> rapports
[2.0273605392933085, 1.9563162857126917, 2.0614220204416864, 1.965353807843951,
2.0106148145645393, 1.9991867362742928, 2.00937735149579]
"""


def u2(n):
    if n == 0:
        return 1.
    prec = u2(n-1)
    return (prec+2/prec)/2

les_temps2 = []
for n in range(15, 18):  # (15, 23)
    t0 = time()
    _ = u2(n)
    t1 = time()
    les_temps2.append(t1 - t0)

"""
>>> les_temps2
[1.9073486328125e−05, 1.2874603271484375e−05, 1.2159347534179688e−05]
```

```
"""

# Exo 5


def catalan(n):
    if n == 0:
        return 1
    return sum(catalan(k)*catalan(n-1-k) for k in range(n))

les_temps = []
for n in range(8, 10):  # (8, 15)
    t0 = time()
    _ = catalan(n)
    t1 = time()
    les_temps.append(t1 - t0)

rapports = [les_temps[k+1]/les_temps[k] for k in range(len(les_temps) - 1)]
"""
>>> les_temps
[0.004868030548095703, 0.014584064483642578, 0.0434870719909668,
0.13451504707336426, 0.39469194412231445, 1.2035799026489258,
3.5697619915008545]
>>> rapports
[2.995885982956215, 2.9818211541605364, 3.093219223894999, 2.934184336322242,
3.049415932026062, 2.9659534723405265]

Le nombre d'appels récursifs vérifie une relation de récurrence qui peut sembler compliquée...
 mais en faisant la différence de deux termes successifs, tout s'arrange !
"""

# Exo 6


def decomposition(n):
    if n <= 1:
        return [n]
    res = decomposition( n // 2 )
    res.append( n % 2 )    # bit de poids fort à droite
    return  res

"""
>>> decomposition(42)
[1, 0, 1, 0, 1, 0]
>>> decomposition(1789)
[1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1]
"""

# Exo 7


def pgcd(a, b):
    if min(a, b) == 0:
        return max(a, b)
    return pgcd(b, a % b)

"""
>>> pgcd(1789, 945)
1
>>> pgcd(1789*42, 945*42)
42
"""
```



```
# Exo 8


def bezout(a, b):
    if a < b:  # je préfère l'autre cas
        (u, v) = bezout(b, a)
        return v, u
    # maintenant, b <= a
    if b == 0:  # le pgcd vaut a, et il y a une relation de Bezout simple...
        return (1, 0)
    (q, r) = (a // b, a % b)     # a=bq+r
    (u1, v1) = bezout(b, r)      # bu1+rv1=pgcd, ou encore bu1+(a-bq)v1=pgcd
                                 # ou encore : av1+b(u1-qv1)=pgcd
    return (v1, u1-q*v1)

"""

>>> bezout(945, 1789)
(248, −131)
>>> 945*248 − 1789*131
1
"""

# Exo 9


def recomposition(t):
    """ retrouver un entier à partir de son écriture en base 2"""
    if len(t) == 1:
        return t[0]
    return t[-1] + 2*recomposition(t[:-1])

"""
>>> recomposition(decomposition(1789))
1789
>>> recomposition(decomposition(945))
945
"""

# Exo 10


def appartient(x, t):
    """ recherche dans une liste triée """
    if t == []:
        return False
    if len(t) == 1:
        return x == t[0]
    milieu = len(t)/2  # décalé vers la droite
    if t[milieu] == x:
        return True    # On pourrait éviter les if, en travaillant un peu...
    if t[milieu] < x:
        return appartient(x, t[milieu+1:])
    return appartient(x, t[: milieu])

"""
>>> appartient(5, list(range(6)))
True
>>> appartient(5, list(range(60)))
True
>>> appartient(500, list(range(60)))
```

```
False
>>> appartient(51, list(range(0, 60, 120)))
False
"""

# Exo 11


def recherche_dicho_entre(x, t, i, j):
    milieu = (i+j+1)//2  # +1 pour conserver le biais à droite !
    if t[milieu] == x:
        return True
    if t[milieu] < x:
        return recherche_dicho_entre(x, t, milieu+1, j)
    return recherche_dicho_entre(x, t, i, milieu-1)


def recherche_dicho_logarithmique(x, t):
    recherche_dicho_entre(x, t, 0, len(t)-1)

# Exo 12


def miroir(s):
    """ miroir d'une chaîne; quadratique """
    if len(s) <= 1:
        return s
    return miroir(s[1:])+s[0]
"""
>>> miroir('PLOUF'), miroir('BARBARA')
('FUOLP', 'ARABRAB')
"""


# Exo 13

def chemins_naif(i, j):
    if i*j == 0:
        return 1
    return chemins_naif(i-1, j) + chemins_naif(i, j-1)

"""
>>> chemins_naif(10, 10)
184756
>>> chemins_naif(15, 10)
3268760
>>> from math import factorial
>>> factorial(20)/factorial(10)**2
184756L
>>> factorial(25)/factorial(10)/factorial(15)
3268760L

Pour (20, 20), ça prend trop de temps (de l'ordre de 10^10 appels récursifs !)
"""

# Exo 14

valeurs_chemins = [[1] * 21] + [[1] + [0]*20 for _ in range(20)]


def chemins_vers(i, j):
    """ nombre de chemins entiers croissants de (0,0) vers (i,j)"""
```



```
    if valeurs_chemins[i][j] != 0:
        return valeurs_chemins[i][j]
    resultat = chemins_vers(i-1, j) + chemins_vers(i, j-1)
    valeurs_chemins[i][j] = resultat
    return resultat
"""
>>> chemins_vers(20, 20)
137846528820L
>>> factorial(40)/factorial(20)**2
137846528820L
"""

# Exo 15

valeurs = [[0, 1] + [-1]*799 for _ in range(801)]


def partitions_majorees(n, m):
    """ partitions de n en somme de machins majorés par m"""
    if m == 0:
        return 0
    if m >= n:
        return 1 + partitions_majorees(n, n-1)
    if valeurs[n][m] != -1:
        return valeurs[n][m]
    resultat = partitions_majorees(n-m, m) + partitions_majorees(n, m-1)
    valeurs[n][m] = resultat
    return resultat


def partitions(n):
    return partitions_majorees(n, n)

"""
>>> partitions(200)
3972999029388L

Valeur effectivement trouvée par Mac Mahon à la main !!

>>> partitions(721)
161061755750279477635534762L

C'est en effet le bon résultat, conjecturé par Lehmer grace à un développement asymptotique
"""


# Partie 4 : problème du cavalier

largeur, hauteur = 6, 6
deplacements = [(2,1), (1,2), (-1,2), (-2,1), (-2,-1), (-1,-2), (1,-2), (2,-1)]

echiquier = [[False]*largeur for _ in range(hauteur)]  # Variable globale
echiquier[0][0] = True  # On part du bas à gauche


def valide(x, y, dx, dy):  # Tel déplacement est-il valide ?
    return x+dx < largeur and x+dx >= 0 and y+dy < hauteur and y+dy >= 0\
            and not(echiquier[x+dx][y+dy])


chemin = [(0, 0)] * largeur * hauteur
nb_back = 0  # On compte le nombre de backtracks
```

```
def backtrack((x, y), places):  # le nombre de places déjà visitées
    global nb_back
    nb_back += 1
    if places == largeur*hauteur:
        return True
    for (dx, dy) in deplacements:
        if valide(x, y, dx, dy):
            echiquier[x+dx][y+dy] = True
            chemin[places] = (x+dx, y+dy)
            if backtrack((x+dx, y+dy), places+1):
                return True
            echiquier[x+dx][y+dy] = False  # On nettoie
    return False

t0 = time()
resultat = backtrack((0, 0), 1)
t1 = time()
if resultat:
    print"Solution :\n"+str(chemin)
else:
    print "Pas de solution"
print "%i backtracks, en %f secondes" % (nb_back, t1-t0)

"""
(4,4) : Pas de solution
2223 backtracks, en 0.012781 secondes

(6,6) : Solution :
[(0, 0), (2, 1), (4, 2), (5, 4), (3, 5), (1, 4), (0, 2), (2, 3), (4, 4),
(2, 5), (0, 4), (1, 2), (2, 4), (0, 5), (1, 3), (0, 1), (2, 0), (4, 1),
(5, 3), (4, 5), (3, 3), (5, 2), (4, 0), (3, 2), (1, 1), (0, 3), (1, 5),
(3, 4), (5, 5), (4, 3), (5, 1), (3, 0), (2, 2), (1, 0), (3, 1), (5, 0)]
248169 backtracks, en 1.682929 secondes

(8,8) : Solution :
[(0, 0), (2, 1), (4, 2), (6, 3), (7, 5), (6, 7), (4, 6), (2, 7), (0, 6),
(1, 4), (3, 5), (5, 6), (7, 7), (6, 5), (5, 7), (3, 6), (1, 7), (0, 5),
(2, 6), (4, 7), (5, 5), (7, 6), (6, 4), (4, 5), (6, 6), (5, 4), (3, 3),
(2, 5), (3, 7), (1, 6), (0, 4), (1, 2), (2, 4), (0, 3), (1, 1), (3, 0),
(2, 2), (1, 0), (0, 2), (2, 3), (4, 4), (3, 2), (4, 0), (6, 1), (7, 3),
(5, 2), (7, 1), (5, 0), (3, 1), (4, 3), (5, 1), (7, 0), (6, 2), (7, 4),
(5, 3), (7, 2), (6, 0), (4, 1), (2, 0), (0, 1), (1, 3), (3, 4), (1, 5), (0, 7)]
8250733 backtracks, en 55.203664 secondes
"""
```