```
# -*- coding: utf-8 -*-
"""
Created on Tue May 30 21:52:10 2017

@author: stephane
"""

from scipy.integrate import odeint              # Exo 1
import matplotlib.pyplot as pypl                # Exo 1
from numpy import linspace, array, array_equal  # Exos 1, 4
from numpy.random import rand                # Exo 3
from random import randint                      # Exo 4


#
# Exercice 1 : proies vs prédateurs
#

a, b, c, d = 2, 0.3, 1.5, 0.5


def F(X, t):
    x, y = X
    return [a*x-b*x*y, -c*y+d*x*y]


les_t = linspace(0, 5, 1000)

for initiale in [[10, 10], [7, 7], [5, 7], [3, 7]]:
    solution = odeint(F, initiale, les_t)
    pypl.plot(solution[:, 0], solution[:, 1])
    pypl.plot([initiale[0]], [initiale[1]], 'ro')

pypl.grid()


pypl.savefig('dessin-proies.pdf')


#
# Exercice 2 : décomposition en base 2
#


def decomp2(n):
    if n < 2:
        return [n]
    return [n % 2] + decomp2(n // 2)


def T(n):
    return sum(decomp2(n))


def decomp2iter(n):
    dec = []
    nc = n
    while nc > 0:
        dec.append(nc % 2)
        nc = nc // 2
    return dec
```



```
def Tp(n):
    return sum(decomp2iter(n))


def T_rec(n):
    if n < 2:
        return n
    return (n % 2) + T_rec(n // 2)


"""
>>> decomp2(1789)
[1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1]
>>> decomp2iter(1789)
[1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1]
>>> T(1789)
9
>>> Tp(1789)
9
>>> T_rec(1789)
9
"""

#
# Exercice 3 : un peu de probas
#

"""
>>> LR = rand(6)
>>> LR
array([ 0.01759413,  0.01157585,  0.69965456,  0.55148205,  0.21542905,
        0.76428978])
>>> LR < 0.5
array([ True,  True, False, False,  True, False], dtype=bool)
>>> 1*(LR < 0.5)
array([1, 1, 0, 0, 1, 0])

Franchement, c'est MISÉRABLE, cette façon de faire.
"""


def tirage(n):
    return 1*(rand(n) < 0.5)

# C'est misérable de stocker ces valeurs juste pour les sommer...


def experience(n):
    return sum(tirage(n)) / float(n)
"""
>>> [experience(10**k) for k in range(2, 8)]
[0.55000000000000004, 0.51500000000000001, 0.49049999999999999
 0.50173999999999996, 0.50005999999999995, 0.49988939999999998]
"""


def attente(motif, tirage):
    lm = len(motif)
    for i in range(len(tirage)-len(motif)+1):
        if tirage[i:i+lm] == motif:
            return i+lm-1
```

```
    return None
"""
>>> attente([1,0,1,1], [0,0,1,0,0,1,0,1,1])
8
"""

#
# Exercice 4 : des sous-matrices
#

dimA, dimB = 9, 2
maxv = 6


def sous_matrice(A, B):
    La, Ca = A.shape
    Lb, Cb = B.shape
    for i in range(La-Lb):
        for j in range(Ca-Cb):
            if array_equal(A[i:i+Lb, j:j+Cb], B):
                return True, i, j
    return False


for _ in range(10):
    A0 = array([[randint(0, maxv) for _ in range(dimA)] for _ in range(dimA)])
    B0 = array([[randint(0, maxv) for _ in range(dimB)] for _ in range(dimB)])
    print(sous_matrice(A0, B0))


def somme_carres(A, B):  # Utilisons à fond numpy ! (attention les yeux)
    return sum(sum(ligne) for ligne in (A-B)*(A-B))

"""
>>> somme_carres(B0, B0)
0
>>> somme_carres(2*B0, B0)
45
>>> B0
array([[6, 3],
    [0, 0]])
"""


def approcher(A, B):
    La, Ca = A.shape
    Lb, Cb = B.shape
    dist, posi, posj = -1, 0, 0
    for i in range(La-Lb):
        for j in range(Ca-Cb):
            if dist < 0 or somme_carres(A[i:i+Lb, j:j+Cb], B) < dist:
                dist, posi, posj = somme_carres(A[i:i+Lb, j:j+Cb], B), i, j
    return dist, A[posi:posi+Lb, posj:posj+Cb]

print(A0)
print(B0)
print(approcher(A0, B0))

"""
[[4 1 4 2 3 4 3 1 1]
 [4 0 0 6 5 0 2 6 1]
 [6 3 5 6 3 1 3 1 6]
```



```
 [4 0 1 2 2 4 4 6 3]
 [3 5 2 5 0 0 1 5 0]
 [6 1 5 5 2 0 6 2 2]
 [1 2 2 6 6 4 1 6 6]
 [4 5 6 0 1 6 2 3 5]
 [5 0 4 3 3 3 2 2 4]]
[[6 3]
 [0 0]]
(6, array([[4, 4],
       [0, 1]]))
"""
```