```python
#Correction du TP de révisions - Révisions I

#EX1
#L'instruction pour connaître l'image de 1 est : s[1]. De la meme façon, la transposition (2 3)
est définie par ses images : t=[0,1,3,2], en veillant à ce que 0 et 1 soient invariants.

def comp(s1:list,s2:list):
    n=len(s1)
    p=len(s2)
    assert n==p
    s=[]
    for i in range(0,n):
        s.append(s2[s1[i]])
    return s

def inv(s1:list):
    n=len(s1)
    s=[0 for i in range(0,n)] #on construit une liste de n images qu'on va compléter au fur et à
mesure
    for i in range(0,n):
        s[s1[i]]=i
    return s

def groupe(G:list):
    n=len(G[0])#on suppose que tous les éléments de G ont le même nombre d'éléments
    #on va tester les conditions donnant un groupe... et renvoyer False dès qu'une condition ne
sera pas remplie, sinon c'est un groupe !
    #élément neutre
    if [i for i in range(0,n)] not in G:
        return False
    else:
        #stabilité par composition
        for s1 in G:
            for s2 in G:
                if comp(s1,s2) not in G:
                    return False
        #stabilité par l'inverse
        for s1 in G:
            if inv(s1) not in G:
                return False
        return True

def cyclique(s:list):
    n=len(s)
    id=[i for i in range(0,n)]# on va comparer les itérées de s avec l'identité... qu'on ajoute
au fur et à mesure.
    G=[id]
    g=s
    while g!=id:
        G.append(g)
        g=comp(s,g)
    return G

#EX2
from numpy import *

def produit(A:array,B:array):
    (n,p)=shape(A)
    (m,q)=shape(B)
    assert p==m
    C=zeros((n,q))#on construit un tableau nul qu'on va compléter à l'aide de la formule du
produit matriciel.
    for i in range(0,n):
        for j in range(0,q):
            S=0
            for k in range(0,p):
                S=S+A[i,k]*B[k,j]
            C[i,j]=S
    return C

def puiss(A:array,n0:int):
    (n,p)=shape(A)
    assert n==p#on teste si on travaille avec une matrice carrée
    if n0==0:
        return eye(n)
    else:
```

```
        P=eye(n)
        for k in range(1,n0+1):
            P=produit(A,P)
        return P

def puissrapide(A:array,n0:int):
    (n,p)=shape(A)
    assert n==p#on teste si on travaille avec une matrice carrée
    if n0==0:
        return eye(n)
    else:
        if n0%2==0:#on teste si l'entier est pair
            return produit(puissrapide(A,n0//2),puissrapide(A,n0//2))
        else:
            return produit(produit(puissrapide(A,(n0-1)//2),puissrapide(A,(n0-1)//2)),A)


#EX3
def diviseurs(n):
    """diviseurs(n:int)->list
    """
    L=[]
    for k in range(1,n+1):
        if n%k==0:
            L.append(k)
        else:
            pass
    return L

def nextprime(n):
    """nextprime(n:int)->int
    """
    p=n+1
    while len(diviseurs(p))>2:
        p=p+1
    return p

def decomposition(n):
    """decomposition(n:int)->list
    """
    assert type(n)==int and n>=2 #wawan et ses préconditions...
    L=[]
    p=2
    while p<=n and len(diviseurs(p))==2:
        #on va alors tester si p divise n, n/p, n/p^2...
        N=n
        while N%p==0:
            L.append(p)
            N=N//p
        p=p+1
    return L

def valuation(p,n):
    """valuation(p:int,n:int)->int
    """
    L=decomposition(n)
    if p not in L:
        return 0
    else:
        #on va compter le nombre de fois que p intervient
        c=0
        for x in L:
            if p==x:
                c=c+1
            else:
                pass
        return c

#EX4
def pointmilieu(a,b,f,n):
    """pointmilieu(a:float,b:float,f:function,n:int->float)
    """
    S=0
    h=(b-a)/n
    for k in range(0,n):
        x=a+k*h
```

```
        y=x+h
        S=S+h*f((x+y)/2)
    return S

#random() renvoie un réel au hasard compris entre 0 et 1.
#La fonction f nous permet d'obtenir un quart de disque...

from math import *
from random import *
def montecarlo(n):
    """montecarlo(n:int)->float
    """
    c=0
    for k in range(0,n):
        x,y=random(),random()
        if y<=sqrt(1-x**2):
            c=c+1
        else:
            pass
    return c/n

#C'est la loi faible des grands nombres : la fréquence de succès tend vers la probabilité d'être
dans ce quart de cercle, soit pi/4.

#EX5
def fibo1(n):
    """fibo1(n:int)->int
    """
    assert type(n)==int and n>=0
    if n==0 or n==1:
        return 1
    else:
        u0,u1=1,1
        for k in range(2,n+1):
            u2=u1+u0
            u0,u1=u1,u2
        return u2

def fibo2(n):
    """fibo2(n:int)->int
    """
    assert type(n)==int and n>=0
    if n==0 or n==1:
        return 1
    else:
        return fibo2(n-1)+fibo2(n-2)

def fibo3(n,D={}):
    """fibom(n:int,D:dict)->int
    """
    assert type(n)==int and n>=0
    if n==0 or n==1:
        return 1
    else: #on va utiliser la relation de récurrence, en vérifiant si les valeurs ont déjà été
calculées
        if n-1 in D:
            c1=D[n-1]
        else:
            c1=fibo3(n-1,D)
        if n-2 in D:
            c2=D[n-2]
        else:
            c2=fibo3(n-2,D)
        c=c1+c2
        D[n]=c #on stocke le résultat avant de renvoyer le calcul
        return D[n]


from pylab import *
from time import *
def comparaison(n):
    """comparaison(n:int)->None
    """
    X=[k for k in range(0,n+1)]
    Y1=[]
    for k in X:
```

```
    t0=time()
    fibo1(k)
    t1=time()
    Y1.append(t1-t0)
Y2=[]
for k in X:
    t0=time()
    fibo2(k)
    t1=time()
    Y2.append(t1-t0)
Y3=[]
for k in X:
    t0=time()
    fibo3(k)
    t1=time()
    Y3.append(t1-t0)
plot(X,Y1,label='fibo1')
plot(X,Y2,label='fibo2')
plot(X,Y3,label='fibo3')
legend()
show()
```