```python
#Correction_info1

#EX1
def racines(a:float,b:float,c:float)->tuple:
    """calcule les racines du trinôme associé
    """
    assert a!=0,'on veut un polynôme du second degré'

    D=b**2-4*a*c #discriminant
    if D>0:
        x1=(-b+D**0.5)/(2*a)
        x2=(-b-D**0.5)/(2*a)
        return x1,x2
    elif D==0:
        x0=-b/(2*a)
        return x0
    else:
        d=1j*(-D)**0.5 #on définit une racine carrée de D
        x1=(-b+d)/(2*a)
        x2=(-b-d)/(2*a)
        return x1,x2

#EX2
def facto(n:int)->int:
    """calcule n!
    """
    P=1
    for k in range(1,n+1):
        P=P*k
    return P

def somme1(n:int)->float:
    """calcule la somme des 1/k!
    """
    S=0
    for k in range(0,n+1):
        S=S+1/facto(k)
    return S

def somme2(n:int)->float:
    """calcule la somme des 1/k!
    """
    S=1
    P=1
    for k in range(1,n+1):
        P=P*k
        S=S+1/P
    return S

from math import *
from time import *
def comparaison(eps:float)->list:
    """comparaison(eps:float)->list
    """
    assert eps>0
```



```python
    t1=time()
    n=0
    while abs(somme1(n)-exp(1))>eps:
        n=n+1
    t2=time()

    t3=time()
    n=0
    while abs(somme2(n)-exp(1))>eps:
        n=n+1
    t4=time()
    return [t2-t1,t4-t3]

#EX3
def u(n:int,u0:int)->int:
    """calcule u_n pour la suite de Syracuse
    """
    if n==0:
        return u0
    else:
    #sinon, on répète le même schéma à savoir : u1 est calculé en
fonction de la parité de u0.
        for k in range(1,n+1):
            if u0%2==0:
                u1=u0//2
            else:
                u1=3*u0+1
            u0=u1 #on n'oublie pas de replacer la valeur calculée
dans u0 pour l'étape d'après.
        return u0

def syraliste(u0:float)->list:
    """renvoie la liste des éléments
    """
    L=[u0]
    n,x=1,u0
    while x!=1:
        x=u(n,u0)
        L.append(x)
        n=n+1
    return L

def recherchemax()->tuple:
    """renvoie le temps de vol maximal et la valeur de u0 associé
    """
    T=[]
    #on va stocker les temps de vol en fonction de u0, de sorte que
T[0] correspondent à u0=2, T[1] à u0=3... puis, on cherche le
maximum.
    for u0 in range(2,1001):
        T.append(len(syraliste(u0)))

    M,ind=T[0],2
    for k in range(1,len(T)):
```

```
        if T[k]>M:
            M,ind=T[k],k
        else:
            pass
    return M,ind+2

#EX4
def moyenne(X:list)->float:
    """calcule la moyenne de X
    """
    n=len(L)
    S=0
    for k in range(0,n):
        S=S+X[k]
    return S/n

def covariance(X:list,Y:list)->float:
    """calcule la covariance de X et Y
    """
    assert len(X)==len(Y)
    n=len(X)
    S=0
    mX,mY=moyenne(X),moyenne(Y)
    for k in range(0,n):
        S=S+(X[k]-mX)*(Y[k]-mY)
    return S/n

def variance(X:list)->float:
    """calcule la variance de X
    """
    return covariance(X,X)

from pylab import *
def reglin(X:list,Y:list)->tuple:
    """calcule les coefficients a et b pour définir la droite de
régression linéaire
    """
    a=covariance(X,Y)/variance(X)
    b=moyenne(Y)-a*moyenne(X)
    U=linspace(min(X),max(X),10)
    V=[a*x+b for x in U]
    plot(X,Y,"*")
    plot(U,V,label="droite de régression linéaire")
    show()
    legend()
    return a,b

#EX5
def chercherliste(L:list)->tuple:
    """renvoie la valeur maximale et son indice
    """
    n=len(L)
    ind=0
    MAX=L[0]
    #on va alors comparer le éléments de la liste à cette valeur de
```



```
référence : on stocke ind,MAX quand le max a été dépassé.
    for k in range(1,n):
        if L[k]>MAX:
            ind=k
            MAX=L[k]
        else:
            pass
    return ind,MAX

from numpy import *
def cherchermatrice(M:array)->tuple:
    """renvoie la valeur maximale et son indice
    """
    p,q=shape(M)
    ind=0,0
    MAX=M[0,0]
    #on va alors comparer le éléments de la matrice à cette valeur de
référence : on stocke ind,MAX quand le max a été dépassé.
    for i in range(0,p):
        for j in range(0,q):
            if M[i,j]>MAX:
                ind=i,j
                MAX=M[i,j]
            else:
                pass
        return ind,MAX
```