```
var('u');
P=(u^2+1)*(u^2-2);
expand(P);#ou P.expand()

-----------------------------------------------------
P.subs(u=2);

-----------------------------------------------------
Q=(x^2+2*x+1)/(x+1);Q

-----------------------------------------------------
Q.simplify() #ne marche pas

-----------------------------------------------------
Q.simplify_rational()

-----------------------------------------------------
R=sqrt(x^2);R

-----------------------------------------------------
R.simplify_rational()#ne marche pas

-----------------------------------------------------
assume(x>0);
R.simplify()

-----------------------------------------------------
def factor_rec(n):
    if n==1:
        return 1
    return n*factor_rec(n-1)
def factor_it(n):
    p=1;
    for i in [1..n]:
        p=p*i
    return p;
%timeit factor_rec(50)

-----------------------------------------------------
%timeit factor_it(50)

-----------------------------------------------------
%timeit factorial(500)

-----------------------------------------------------
L=[1..100];
f(x)=2*x+1;
K=map(f,L);
s=0;
for i in [0..99]:
    s=s+L[i]*K[i]
s
```

```
-------------------------------------------------
def prodscal(L1,L2):
    #cette fonction calcule le produit scalaire de 2 listes
    #on traite d'abord les cas particuliers
    if len(L1)!=len(L2):
        print('les deux listes n\'ont pas la même taille, produit
scalaire impossible');
    elif len(L1)==0:
        print('les listes sont vides, le produit scalaire n\'a pas
de sens')
    else:
        #on traite ensuite le cas général
        s=0;
        for i in [0..len(L1)-1]:
            s=s+L1[i]*L2[i]
        return s
prodscal([],[]);
prodscal([],[1]);
prodscal(L,K)

-------------------------------------------------
L1=[];
for i in [1..100]:
    if is_prime(i):
        L1.append(i)
print(L1);

-------------------------------------------------
L2=filter(is_prime,[1..100]);
print(L2);

-------------------------------------------------
L3=[];
i=2;
while i <=100:
    L3.append(i);
    i=next_prime(i);
L3

-------------------------------------------------
prod(L1);

-------------------------------------------------
def syracuse(un):
    if un%2 == 0:
        return un//2
    else:
        return 3*un+1

L=[17];
for i in [1..19]:
    L.append(syracuse(L[i-1]))
L
```

```
-----------------------------------------------------
def conjecture(u0):
    un=u0;
    cmpt=0;
    while un != 1:
        un=syracuse(un)
        cmpt=cmpt+1
    return cmpt
conjecture(17)

-----------------------------------------------------
S=0;
for i in [1..100]:
    alea=ZZ.random_element(1000)
    S=S+conjecture(alea+1)
(S+0.)/100

-----------------------------------------------------
P=x^4+3*x^2+2;
factor(P)

-----------------------------------------------------
QX.<X>=PolynomialRing(QQ,'X');#ou QX.X=QQ['X']
P=X^4+3*X^2+2;
factor(P)

-----------------------------------------------------
FpX.<X>=PolynomialRing(GF(47),'X');
P=X^4+3*X^2+2;
factor(P)

-----------------------------------------------------
FpX.<X>=PolynomialRing(GF(43),'X');
P=X^4+3*X^2+2;
factor(P)

-----------------------------------------------------
FpX.<X>=PolynomialRing(GF(41),'X');
P=X^4+3*X^2+2;
factor(P)

-----------------------------------------------------
def pol2list(P):
    n=P.degree();
    L=[0..n];
    for i in L:
        L[i]=P[i]
    return L

-----------------------------------------------------
pol2list(P)

-----------------------------------------------------
```

```
def deriv(P):
    L=pol2list(P);
    dL=[];
    for i in [1..len(L)-1]:
        dL.append(i*L[i])
    dP=0;
    for i in [0..len(dL)-1]:
        dP=dP+dL[i]*P.parent().gen()^i
    return dP
deriv(P);

-------------------------------------------------
ZT.<T>=ZZ['T'];
deriv(T^3)

-------------------------------------------------
def is_racines_simples(P):
    dP=deriv(P);
    if gcd(P,dP)==1:
        return True
    return False

-------------------------------------------------
is_racines_simples(P)

-------------------------------------------------
PP=(T-6)*(T+5)*(T+1);PP

-------------------------------------------------
is_racines_simples(PP)

-------------------------------------------------
PP11=PP.change_ring(GF(11))

-------------------------------------------------
PP11

-------------------------------------------------
is_racines_simples(PP11)

-------------------------------------------------
def prs(L,K):
    if K==[] or L==[ ] : return 0
    else:
        ps=0;
        m=min(len(K),len(L));
        for i in [0..m-1]: ps=ps+K[i]*L[i]
    return ps

-------------------------------------------------

prs([1,3],[5,7])
```