Affine space, points and vectors

```
K=QQ
L.<a>=FiniteField(q)
K=SR
R=AA
Symbolize(S,size='element,list,
              matrix,polynomial')
view(a)
latex(a)
A,E=AffineSpace(K,d)
P=A([0,-1,2])
u=E([1,1,-1])
K=E.base_field()
d=E.dimension()
Q=P+u
PQ=Arrow(P,Q)
```

Affine varieties

```
Variety(P,E.subspace([u,v]))
BasePoint(X)
Direction(X)
AffineDimension(X)
AffineCodimension(X)
PointAsVariety(X)
Belongs(P,X)
Contained(X,Y)
EqualityOfVarieties(X,Y)
Parallel(X,Y)
AffineSum([X,Y,Z,...])
Meet(X,Y)
```

Affine frames

```
Frame(P,B)
AffineCoordinates(P,Ref)
CanonicalFrame(A)
FrameAsMatrix(Ref)
ChangeOfFrameMatrix(Ref1,Ref2)
ChangeOfFramePoint(P,Ref1,Ref2)
ChangeOfFrameVariety(X,Ref)
```

Equations

```
Poly.<x,y,z,t>=PolynomialRing(QQ,4)
CartesianEquations(X,Poly)
VarietyByEquations(S,A)
```

Miscellaneous

```
Barycenter(S)
Ratio(A,B,C)
```

Affinities

```
f=Affinity(A,Q,h)
f(P)
AffinityAsMatrix(f)
MatrixAsAffinity(M,A)
EquationsOfAffinity(f,Poly)
AffinityByEquations(S,A)
LinearPartOfAffinity(f)
IsBijective(f)
CompositionOfAffinities(f,g)
EqualityOfAffinities(f,g)
InverseOfAffinity(f)
ImageOfVariety(f,X)
Conjugate(f,g)
```

Fixed points and invariant lines

```
FixedPoints(f)
IsInvariant(X,f)
SparseInvariantLines(f)
SpecialInvariantLines(f)
```

Some types of affinities

```
AffineTranslation(v)
AffineHomothecy(r,P)
Symmetry(X,G)
AffineProjection(X,G)
IsTranslation(f)
IsHomothecy(f)
IsSymmetry(f)
IsAffineProjection(f)
RandomTranslation(A)
RandomHomothecy(A)
RandomSymmetry(A,d)
RandomAffineProjection(A,d)
```

Classification of affinities

```
InvarianceLevel(f)
IsSimilar(f,g)
Name(f)
IsSimilar(f,g,transformation=True)
NormalFormOfAffinity(f)
NormalFormOfAffinity(f,transformation=True)
```

Euclidean Affine Spaces

```
AffineSpace(K,d,metric=M)
SquareDistance(X,Y,points=True)
OrthogonalVariety(P,X)
AreOrthogonal(X,Y)
OrthogonalSymmetry(X)
OrthogonalProjection(X)
```

Euclidean motions

```
IsEuclideanMotion(f,orthobase=True)
GlideVector(f)
IsEuclideanSimilar(f,g,transformation=True)
EuclideanNormalFormOfAffinity(f,
              transformation=True)
RotationMatrix(P,theta)
IsRotation(M,params=True)
```