WIT functions	
Function signature	**Description**
from wit:	
resultant(f,g, var = None)	Gives the resultant of the polynomials f and g with respect to var, which by default is the last variable of the the variables of f and g.
discriminant(f,var=None)	Gives the discriminant of the polynomial f with respect to var, which by default is the last variable of the variables of f.
resultant_ext(f,g,var = None)	Yields a list [R,A,B] where R = resultant(f,g,var) and A, B are polynomials such that R = A*f + B*g
imult(f,g,O=[0,0])	Computes the intersecton multiplicity of the curves f=0 and g=0 (f,g bivariate polynomials) at the point O.
homogenize(f,z='')	Homogenizes f using the supplied variable z, which by default is z.
imultinfinity(F,G,O) himult=imultinfinity	If F and G are homogeneous in 3 variables, it computes the intersection multiplicity of the curves F=0 and G=0 at the projective point O. Otherwise, if F and G are affine polynomias, it computes the himult of the homogenization of F and G at the point O, which can be affine or projective.
from wit_power:	
witdual(c)	If c=[c1,c2,c3,...], it returns [-c1,c2,-c3,...]
vprod(x,y,d) vprod(x,y)	If x and y are lists or vectors, it computes p = ([1]+x)*([1]+y) as if the factors were polynomials and returns d terms p[1:], with d = len(x)+len(y) by default.
vpower(x,m,d) vpower(x,m)	If x is a list or vector, it computes p = ([1]+x)**m as if [1]+x were polynomial and returns d terms of p[1:], with d = m*len(x) by default.
invert_vector(c, r) invert_vector(c)	Given list or vector [c1,c2,...,cn], it computes the list or vector [s1,s2,..,sr] such that 1+s1*t+s2*t**2+...+sr t**r is inverse of 1+c1*t+c2*t**2+...+cr t**r mod t**(r+1). By default r = len(c).
todd_numbers(n)	Taylor coeffs of x/(1-e^{-x}) at 0: 1/2, 1/12, 0, -1/720, 0, 1/30240, 0, -1/1209600, 0, 1/47900160, 0, -691/1307674368000, 0, 1/74724249600, 0, -3617 /10670622842880000, 0, 43867/5109094217170944000, 0, -174611/802857662698291200000
bernoulli_numbers(N) BV_ = bernoulli_numbers bernoulli_number(N) B_=bernoulli_number	bernoulli_numbers(N) returns the first N+1 Bernoulli numbers, starting with 1, while bernoulli_number(N) given the N-th Bernoulli number.
bernoulli_polynomial(m,x='x')	Delivers the m-th Bernoulli polynomial BP_m(x) (it has degree m+1) in the variable x.
high_bernoulli_numbers(N,k) HBs_=high_bernoulli_numbers high_bernoulli_number(N,k)	The first N order k Bernoulli numbers and the N-th order k Bernoulli number.
zhe(n,j)	Atiyah's number.
s2n(s,,r='') n2s(n,r='')	For the elementary symmetric polynomials (s) and the Newton sums (n) of the variables [x1,x2,...], s2n expresses the n's in terms of the s's and n2s goes the other way around. See Fulton, p.56. Examples: s2n([s1,s2])=[s1,s1^2-2s2], n2s([n1,n2])=[n1,(n1^2-n2)/2].
c2p(c,r='') p2c(p,r='')	If c=[c1,c2,...] and [n1,n2,...] are the elementary symmetric polynomials and the Newton sums of the variables [x1,x2,...], c2p maps c to p=[n1/1!, n2/2!,...] and p2c goes the other way around. c2p(c:Vector):= c2p(c,length(c)).
monomial(X,E)	Monomial on a list or vector of expressions X with given exponents E.

WIT functions	
Function signature	**Description**
partitions(n,d)	Given a list or vector d of positive interger, and an integer n, find the list of tuples of non-negative integers [m1,...,mr] such that m1*d1+...+mr*dr=n (partitions of n with weights d1,...,dr).
monomial_list(X,d,n)	Given variables x={x1,x2,...,xr} of degrees d={d1,d2,...,dr}, get the list of monomials x1^m1 ... xr^mr such that m1*d1+...+mr*dr=n. The list of the lists {m1,...,mr} is returned by partitions(d,n).
symmetric_polynomial(x,k) symmetric_polynomials(x,K='')	Elementary symmetric polynomial of degree k in the expressions x. Vector of elementary symmetric polynomials of degree k in K for the expressions x.
newton_sum(x,k) newton_sums(x,K)	Newton sum of degree k for the expressions x Vector of Newton sums for the expressions x of degree k in K
stirling_numbers(n) stirling_number_1st(n,k) unsigned_stirling_number_1st(n,k) stirling_number_2nd(n,k)	Returns the list of signed Stirling numbers of the first kind of order n. Returns the k-th signed Stirling number of order n. Returns the k-th unsighed Stirling number of order n. Returns the k-th Stirling number of the second kind of order n.
wdeg(p,w)	Degree of the multivariate polynomial p assuming that its variables have weights w.
chpad(v,r)	Padded form of the Chern characters vector of the Chern polynomial v.
from wit_var_sheaf:	
SH(r, c, d=None, name = None)	Generic constructor of a Sheaf of rank r and chern character c; d is a truncating index; can be given a name using the 4th parameter.
sheaf(r,c,d='', name = '')	
bundle(n,c,d='',name='')	Vector bundle of rank n with Chern classes c = [c1,c2,...] on variety X
trivial_bundle(n,d='',name='')	Creates the trivial bundle of rank n.
o_(d,k='',name='')	The line bundle O(d).
O_(d='', name='')	
adams(k, x)	The k-th Adams operator acting on x
ch(F) rk(F)	Thes functions supply the chern character and rank of F, respectively.
chern_character(F,T) change_ch_2_size(F,k)	
chern_vector(F, k = '') chern(F, k, d = None)	
segre_vector(F, d = None) segre(F, k, d = None)	
todd_vector(F, d = None) todd(F, k) todd_character(F, T) Td	

WIT functions	
Function signature	**Description**
Hom(F,G, k = '') End(F) hom = Hom	
wedge(F, p, d = '') symm(k,E) koszul(F)	
from wit_mor:	
pairing(p,T,v)	
codimension(x,X)	
vector_bundle(P)	
fiber_dim(P)	
base_dim(P)	
lowerdata(P)	
lowerstar(f,x)	
section(P)	
projective_bundle(X,E,h='h',y=False, name = '')	
inclusion(B)	
cl(B)	
bundle_section(X,F, name = '')	
## Class Blowup and its functions	
blowup_locus(W)	
from wit_chow:	
chi(X,F,T='T')	Euler characteristic of the sheaf F as a polynomial in T
HRR(X,F,T='T')	The Hirzebruch-Riemann-Roch theorem of the sheaf F on the space X as a polynomial in T

WIT functions	
Function signature	**Description**
from wit_lie:	**See witlets/wit_lie.ipynb. Refs: 2009-RichterGeber--Geometriekalküle 2008-Cecil--Lie Sphere Geometry With Applications to Submanifolds 2012-Benz--Classical Geometries in Modern Contexts Geometry of Real Inner Product Spaces 2018-Kisil--Lectures on Moebius-Lie Geometry and its Extension**
is_pair = ispair	is_pair(x) is True when x is a pair (x=(a,b)), otherwise False.
Lie_vector(M=(0,0),R=0) LV = lie_vector=Lie_vector	Lie vector of the circle with center [Mittlepunkt] M and radius R in the Euclidean plane. For nonzero R, the orientation of the circle is encoded as the sign of R. Points are encoded as circles of radius 0. By default, R=0, so it gives the Lie vector of M, which by default is (0,0). For the Lie vector of a line, M = (a,b) has to be a normal vector and R = (u,v) a point on the line. The line is oriented by the direction vector (-b,a).
orientation(X)	If X is the Lie vector of circle, this expression gives the sign of R.
Lie_metric(X,Y) LM = lie_metric = Lie_metric	
Lie_form(X) LF = lie_form = Lie_form	
lie_gram_matrix(*S)	
is_lie_vector(X)	
lie_type(X) LT = Lie_type = lie_type	
is_point(X)	
is_line(X)	
is_circle(X)	
normalize(A)	
reorient(A) mirror = reorient	
# Back from Lie objects to Euclidean objects	
circle(X)	
line(X)	
point(X)	
Lie_angular_metric(X,Y) LAM = lie_angular_metric = Lie_angular_metric	
crossing_type(X,Y)	
crossing_angle(X,Y)	
solve_quadratic(a,b,c)	

WIT functions	
Function signature	**Description**
Lie_section(A,B,C) lie_section = Lie_section	