Sage Vorlesung 1

```
# https://cloud.sagemath.com
# doc.sagemath.org/html/en/tutorial
```

```
4
```

```
# Arithmetik
```

```
2+2
```

```
4
```

```
2^3; 2**3;
```

```
8
8
```

```
10%3 # 10 Modulo 3
```

```
1
```

```
10 // 3 # ganzzahliger Quotient
```

```
3
```

```
3^2*4+2%5
```

```
38
```

```
type(1)
```

```
<type 'sage.rings.integer.Integer'>
```

```
pi;e;
```

```
pi
e
```

```
n(pi);n(e); # liefert eine numerische Approximation
```

```
3.14159265358979
2.71828182845905
```

```
type(pi);type(n(pi));
```

```
<type 'sage.symbolic.expression.Expression'>
<type 'sage.rings.real_mpfr.RealNumber'>
```

```
pi.n(digits=14) # auf 14 Stellen genau
```

```
3.1415926535898
```

```
2.5*2 # Dezimalzahl
```

```
5.00000000000000
```

```
5/2*2; 5/3*2; type(5/3) # rationale Zahl
```

```
5
10/3
<type 'sage.rings.rational.Rational'>
```

```
exp(2); e^2;
```

```
e^2
```

```
e^2
```

```
n(exp(2))
```

```
7.38905609893065
```

```
exp(2).n(digits=10); exp(2).n(digits=200)
```

```
7.389056099
7.38905609893065022723042746057500781318031557055184732408712782252257379607905776338431248507912179477375316126547886612388460369278127337447839221339807777490012289560741075370239133094755068208658185
```

```
exp(-infinity)
```

```
0
```

```
ln(e); log(e); log(1); log(2);
```

```
1
1
0
log(2)
```

```
log(10,2) # log von 10 zur Basis 2
```

```
log(10)/log(2)
```

```
n(_); n(log(10,2));
```

```
3.32192809488736
3.32192809488736
```

```
# Die Hilfe aufrufen
```

```
log?
```

File: /home/sage/sage-5.1/local/lib/python2.7/site-packages/sage/functions/log.py

Type: <type 'function'>

Definition: log(x, base=None)

Docstring:

Return the logarithm of x to the given base.

Calls the `log` method of the object x when computing the logarithm, thus allowing use of logarithm on any object containing a `log` method. In other words, log works on more than just real numbers.

EXAMPLES:

```
sage: log(e^2)
2
sage: log(1024, 2); RDF(log(1024, 2))
10
10.0
sage: log(10, 4); RDF(log(10, 4))
log(10)/log(4)
1.66096404744
```

```
sage: log(10, 2)
log(10)/log(2)
sage: n(log(10, 2))
3.32192809488736
sage: log(10, e)
log(10)
sage: n(log(10, e))
2.30258509299405
```

The log function works for negative numbers, complex numbers, and symbolic numbers too,

picking the branch with angle between and pi πi:

```
sage: log(-1+0*I)
I*pi
sage: log(CC(-1))
3.14159265358979*I
sage: log(-1.0)
3.14159265358979*I
```

For input zero, the following behavior occurs:

```
sage: log(0)
-Infinity
sage: log(CC(0))
-infinity
sage: log(0.0)
-infinity
```

The log function also works in finite fields as long as the argument lies in the multiplicative group generated by the base:

```
sage: F = GF(13); g = F.multiplicative_generator(); g
2
sage: a = F(8)
sage: log(a,g); g^log(a,g)
3
8
sage: log(a,3)
Traceback (click to the left of this block for traceback)
...
```

```
exp(x^2+log(x))
```

```
e^(x^2 + log(x))
```

```
_.simplify()
```

```
x*e^(x^2)
```

```
n(tan(3))
```

```
-0.142546543074278
```

```
abs(-2) # Betrag
```

```
2
```

```
ceil(3.44); floor(3.44); round(3.44);
```

```
4
3
3
```

```
diff(exp(2*x),x)
```
```
    2*e^(2*x)
```
```
diff((1+x)^x,x)
```
```
    (x/(x + 1) + log(x + 1))*(x + 1)^x
```
```
# Aussagen / Logik
```

```
true
```
```
    True
```
```
type(true)
```
```
    <type 'bool'>
```
```
false
```
```
    False
```
```
is_even(2)
```
```
    True
```
```
is_odd(2)
```
```
    False
```
```
6%2==0;
```
```
    True
```
```
2<9
```
```
    True
```
```
true | false
```
```
    True
```
```
true & false
```
```
    False
```
```
import sage.logic.propcalc as propcalc
```

```
propcalc?
```

File: /home/sage/sage-5.1/local/lib/python2.7/site-packages/sage/logic/propcalc.py

Type: <type 'module'>

Definition: propcalc([noargspec])

Docstring:

Propositional Calculus

Formulas consist of the following operators:

- & – and
- | – or
- ~ – not
- ^ – xor
- -> – if-then

- <-> – if and only if

Operators can be applied to variables that consist of a leading letter and trailing underscores and alphanumerics. Parentheses may be used to explicitly show order of operation.

AUTHORS:

- Chris Gorecki – propcalc, boolformula, logictable, logicparser, booleval
- Michael Greenberg – boolopt

EXAMPLES:

```
sage: import sage.logic.propcalc as propcalc
sage: f = propcalc.formula("a&((b|c)^a->c)<->b")
sage: g = propcalc.formula("boolean<->algebra")
sage: (f&~g).ifthen(f)
((a&((b|c)^a->c)<->b)&(~(boolean<->algebra)))->(a&((b|c)^a->c)<->b)
```

We can create a truth table from a formula:

```
sage: f.truthtable()
a      b      c      value
False  False  False  True
False  False  True   True
False  True   False  False
False  True   True   False
True   False  False  True
True   False  True   False
True   True   False  True
True   True   True   True
sage: f.truthtable(end=3)
a      b      c      value
False  False  False  True
False  False  True   True
False  True   False  False
sage: f.truthtable(start=4)
a      b      c      value
True   False  False  True
True   False  True   False
True   True   False  True
True   True   True   True
sage: propcalc.formula("a").truthtable()
a      value
False  False
True   True
```

Now we can evaluate the formula for a given set of input:

```
sage: f.evaluate({'a':True, 'b':False, 'c':True})
False
sage: f.evaluate({'a':False, 'b':False, 'c':True})
True
```

And we can convert a boolean formula to conjunctive normal form:

```
sage: f.convert_cnf_table()
sage: f
(a|~b|c)&(a|~b|~c)&(~a|b|~c)
sage: f.convert_cnf_recur()
sage: f
(a|~b|c)&(a|~b|~c)&(~a|b|~c)
```

Or determine if an expression is satisfiable, a contradiction, or a tautology:

```
sage: f = propcalc.formula("a|b")
sage: f.is_satisfiable()
True
sage: f = f & ~f
sage: f.is_satisfiable()
False
sage: f.is_contradiction()
True
sage: f = f | ~f
sage: f.is_tautology()
True
```

The equality operator compares semantic equivalence:

```
sage: f = propcalc.formula("(a|b)&c")
sage: g = propcalc.formula("c&(b|a)")
sage: f == g
True
sage: g = propcalc.formula("a|b&c")
sage: f == g
False
```

TESTS:

It is an error to create a formula with bad syntax:

```
sage: propcalc.formula("")
Traceback (click to the left of this block for traceback)
...
```

```
f=propcalc.formula("(a|b)&c|d&a")
g=propcalc.formula("c&(b|a)|d&c")
f == g
```

```
False
```

```
f = propcalc.formula("a&((b|c)^a->c)<->b")
f.truthtable()
```

a	b	c	value
False	False	False	True
False	False	True	True
False	True	False	False
False	True	True	False
True	False	False	True
True	False	True	False
True	True	False	True
True	True	True	True

```
# Variablen
```

```
a=5
```

```
a
```

```
5
```

```
a==5
```
```
True
```
```
del(a)
```

```
a==5
```
```
Traceback (click to the left of this block for traceback)
...
NameError: name 'a' is not defined
```
```
a=5
```

```
b=3
```

```
b<a
```
```
True
```
```
a+b;
```
```
8
```
```
c=a+b; c
```
```
8
```
```
a!=5; a<>5; #Ungleichheit
```
```
False
False
```
```
(3>5) or (a>4)
```
```
True
```
```
(x<5).negation()
```
```
x >= 5
```
```
# undefinierte Variablen
```

```
x # x ist automatisch definiert
```
```
x
```
```
y=var('y'); y
```
```
y
```
```
reset()
```

```
type(x);
```
```
<type 'sage.symbolic.expression.Expression'>
```
```
a,b,c=var('a,b,c');
(3*a+5*b)+(6*a+7*c)
```
```
9*a + 5*b + 7*c
```
```
s=(sin(x)^2+cos(x)^2);s;
```

```
    sin(x)^2 + cos(x)^2
```

```
s.simplify()
```

```
    sin(x)^2 + cos(x)^2
```

```
s.simplify_trig()
```

```
    1
```

```
p=(3*x^2+7*x+5); q=(7*x^3+5*x-4); p*q;
```

```
    (3*x^2 + 7*x + 5)*(7*x^3 + 5*x - 4)
```

```
expand(_) # ausmultiplizieren
```

```
    21*x^5 + 49*x^4 + 50*x^3 + 23*x^2 - 3*x - 20
```

```
factor(_) # faktorisieren
```

```
    (3*x^2 + 7*x + 5)*(7*x^3 + 5*x - 4)
```

```
# Mengen
```

```
M1=set([1,5,3,2])
```

```
M2=set([1,6,7,2])
```

```
M1;M2; type(M1);
```

```
    set([1, 2, 3, 5])
    set([1, 2, 6, 7])
    <type 'set'>
```

```
M1.union(M2) # Vereinigung
```

```
    set([1, 2, 3, 5, 6, 7])
```

```
M1.intersection(M2) # Schnitt
```

```
    set([1, 2])
```

```
6 in M2
```

```
    True
```

```
9 in M2
```

```
    False
```

```
M1==M2
```

```
    False
```

```
# Vektoren und Matrizen
```

```
a=vector([1,2,3])
```

```
a
```

```
    (1, 2, 3)
```

```
b=vector([4,5,6]);
```

```
a+b
```
```
(5, 7, 9)
```
```
a.get(0)
```
```
1
```
```
a[0]
```
```
1
```
```
c=2.1;
```

```
a*c;a*b;
```
```
(2.10000000000000, 4.20000000000000, 6.30000000000000)
32
```
```
d=a.outer_product(b); d;
```
```
[ 4  5  6]
[ 8 10 12]
[12 15 18]
```
```
M=Matrix([[1,2,3],[4,5,6],[7,8,9]])
```

```
M; M[1];M[1,1];
```
```
[1 2 3]
[4 5 6]
[7 8 9]
(4, 5, 6)
5
```
```
M[0,0]=2;
```

```
M[0,0]
```
```
2
```
```
M[0,:]=2;M;
```
```
[2 2 2]
[4 5 6]
[7 8 9]
```
```
# Lineare Gleichungssystem lösen
```

```
N=matrix(3,4); N;
```
```
[0 0 0 0]
[0 0 0 0]
[0 0 0 0]
```
```
N[:,3]=a; N;
```
```
[0 0 0 1]
[0 0 0 2]
[0 0 0 3]
```

```
N[:,range(0,3)]=M;N;
```

```
[2 2 2 1]
[4 5 6 2]
[7 8 9 3]
```

```
rank(N); rank(M);
```

```
3
2
```

```
x=M.solve_right(a) # löse Gleichung Mx=a
```

```
Traceback (click to the left of this block for traceback)
...
ValueError: matrix equation has no solutions
```

```
M[0,0]=1;M
```

```
[1 2 2]
[4 5 6]
[7 8 9]
```

```
x
```

```
x
```

```
M\a; x=M.solve_right(a);
```

```
(-1/3, 2/3, 0)
```

```
M*x
```

```
(1, 2, 3)
```

```
# Strings
```

```
a='hello'
```

```
a
```

```
'hello'
```

```
b='world'
```

```
a+b
```

```
'helloworld'
```

```
a+' '+b
```

```
'hello world'
```

```
type(a)
```

```
<type 'str'>
```

```
a+1
```

```
Traceback (click to the left of this block for traceback)
...
TypeError: unsupported operand parent(s) for '+': '<type
'str'>' and 'Integer Ring'
```

```
a+'1'
```
```
    'hello1'
```
```
# Listen
```
```
v=[2,3,5]
```
```
v; type(v);
```
```
    [2, 3, 5]
    <type 'list'>
```
```
v=v+['hallo'];v;
```
```
    [2, 3, 5, 'hallo']
```
```
w=vector(v);
```
```
    Traceback (click to the left of this block for traceback)
    ...
    TypeError: unable to find a common ring for all elements
```
```
L=range(1,15);L;
```
```
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```
```
type(L)
```
```
    <type 'list'>
```
```
L=[2^n for n in range (1,10)];L;
```
```
    [2, 4, 8, 16, 32, 64, 128, 256, 512]
```
```
M=set(L);M;
```
```
    set([64, 32, 2, 4, 8, 128, 256, 16, 512])
```
```
# Rechenbeispiele
```
```
n,i=var('n,i')
```
```
sum(i,i,1,n)
```
```
    1/2*n^2 + 1/2*n
```
```
sum(i^5,i,1,10)
```
```
    220825
```
```
x1,x2=var('x1,x2')
```
```
solve([x1^2==-1],x1)
```
```
    [x1 == -I, x1 == I]
```
```
solve?
```

File: /home/sage/sage-5.1/local/lib/python2.7/site-packages/sage/symbolic/relation.py

Type: <type 'function'>

Definition: solve(f, *args, **kwds)

Docstring:

Algebraically solve an equation or system of equations (over the complex numbers) for given variables. Inequ
also supported.

INPUT:

- `f` - equation or system of equations (given by a list or tuple)
- `*args` - variables to solve for.
- `solution_dict` - bool (default: False); if True or non-zero, return a list of dictionaries containing the
return an empty list (rather than a list containing an empty dictionary). Likewise, if there's only a sing
dictionary with that solution.

EXAMPLES:

```
sage: x, y = var('x, y')
sage: solve([x+y==6, x-y==4], x, y)
[[x == 5, y == 1]]
sage: solve([x^2+y^2 == 1, y^2 == x^3 + x + 1], x, y)
[[x == -1/2*I*sqrt(3) - 1/2, y == -sqrt(-1/2*I*sqrt(3) + 3/2)],
 [x == -1/2*I*sqrt(3) - 1/2, y == sqrt(-1/2*I*sqrt(3) + 3/2)],
 [x == 1/2*I*sqrt(3) - 1/2, y == -sqrt(1/2*I*sqrt(3) + 3/2)],
 [x == 1/2*I*sqrt(3) - 1/2, y == sqrt(1/2*I*sqrt(3) + 3/2)],
 [x == 0, y == -1],
 [x == 0, y == 1]]
sage: solve([sqrt(x) + sqrt(y) == 5, x + y == 10], x, y)
[[x == -5/2*I*sqrt(5) + 5, y == 5/2*I*sqrt(5) + 5], [x == 5/2*I*sqrt(5) + 5,
sage: solutions=solve([x^2+y^2 == 1, y^2 == x^3 + x + 1], x, y, solution_dict
sage: for solution in solutions: print solution[x].n(digits=3), ",", solution
-0.500 - 0.866*I , -1.27 + 0.341*I
-0.500 - 0.866*I , 1.27 - 0.341*I
-0.500 + 0.866*I , -1.27 - 0.341*I
-0.500 + 0.866*I , 1.27 + 0.341*I
0.000 , -1.00
0.000 , 1.00
```

Whenever possible, answers will be symbolic, but with systems of equations, at times approximations will b
in Maxima:

```
sage: sols = solve([x^3==y,y^2==x],[x,y]); sols[-1], sols[0]
([x == 0, y == 0], [x == (0.309016994375 + 0.951056516295*I),  y == (-0.80901
sage: sols[0][0].rhs().pyobject().parent()
Complex Double Field
```

If `f` is only one equation or expression, we use the solve method for symbolic expressions, which defaults to

```
sage: solve([y^6==y],y)
[y == e^(2/5*I*pi), y == e^(4/5*I*pi), y == e^(-4/5*I*pi), y == e^(-2/5*I*pi)
sage: solve( [y^6 == y], y)==solve( y^6 == y, y)
True
```

Note

For more details about solving a single equations, see the documentation for its solve.

```
sage: from sage.symbolic.expression import Expression
sage: Expression.solve(x^2==1,x)
[x == -1, x == 1]
```

We must solve with respect to actual variables:

```
sage: z = 5
sage: solve([8*z + y == 3, -z +7*y == 0], y, z)
```

```
sage: solve([0.2 * y == 3, -2.1 * y == 0],y,2)
Traceback (click to the left of this block for traceback)
...
```

```
sudoku?
```

File: /home/sage/sage-5.1/local/lib/python2.7/site-packages/sage/games/sudoku.py

Type: <type 'function'>

Definition: sudoku(m)

Docstring:

Solves Sudoku puzzles described by matrices.

INPUT:

- `m` - a square Sage matrix over $\mathbb{Z}$ where zeros are blank entries

OUTPUT:

A Sage matrix over $\mathbb{Z}$ containing the first solution found, otherwise `None`.

This function matches the behavior of the prior Sudoku solver and is included only to replicate that behavior.

EXAMPLE:

An example that was used in previous doctests.

```
sage: A = matrix(ZZ,9,[5,0,0, 0,8,0, 0,4,9, 0,0,0, 5,0,0, 0,3,0, 0,6,7, 3,0,0
sage: A
[5 0 0 0 8 0 0 4 9]
[0 0 0 5 0 0 0 3 0]
[0 6 7 3 0 0 0 0 1]
[1 5 0 0 0 0 0 0 0]
[0 0 0 2 0 8 0 0 0]
[0 0 0 0 0 0 0 1 8]
[7 0 0 0 0 4 1 5 0]
[0 3 0 0 0 2 0 0 0]
[4 9 0 0 5 0 0 0 3]
sage: sudoku(A)
[5 1 3 6 8 7 2 4 9]
[8 4 9 5 2 1 6 3 7]
[2 6 7 3 4 9 5 8 1]
[1 5 8 4 6 3 9 7 2]
[9 7 4 2 1 8 3 6 5]
[3 2 6 7 9 5 4 1 8]
[7 8 2 9 3 4 1 5 6]
[6 3 5 1 7 2 8 9 4]
[4 9 1 8 5 6 7 2 3]
```

Using inputs that are possible with the `Sudoku` class, other than a matrix, will cause an error.

```
sage: sudoku('.4..32....14..3.')
Traceback (click to the left of this block for traceback)

...
```

```
A = matrix(ZZ,9,[5,0,0,
0,8,0, 0,4,9, 0,0,0, 5,0,0, 0,3,0, 0,6,7, 3,0,0, 0,0,1, 1,5,0, 0,0,0,
0,0,0,  0,0,0, 2,0,8, 0,0,0, 0,0,0, 0,0,0, 0,1,8, 7,0,0, 0,0,4,
1,5,0,
0,3,0, 0,0,2, 0,0,0, 4,9,0, 0,5,0, 0,0,3])
```

```
A;
```

```
[5 0 0 0 8 0 0 4 9]
[0 0 0 5 0 0 0 3 0]
[0 6 7 3 0 0 0 0 1]
[1 5 0 0 0 0 0 0 0]
[0 0 0 2 0 8 0 0 0]
[0 0 0 0 0 0 0 1 8]
[7 0 0 0 0 4 1 5 0]
[0 3 0 0 0 2 0 0 0]
[4 9 0 0 5 0 0 0 3]
```

```
sudoku(A)
```

```
[5 1 3 6 8 7 2 4 9]
[8 4 9 5 2 1 6 3 7]
[2 6 7 3 4 9 5 8 1]
[1 5 8 4 6 3 9 7 2]
[9 7 4 2 1 8 3 6 5]
[3 2 6 7 9 5 4 1 8]
[7 8 2 9 3 4 1 5 6]
[6 3 5 1 7 2 8 9 4]
[4 9 1 8 5 6 7 2 3]
```