Sage Vorlesung 1

```
# https://cloud.sagemath.com     (Sage Online)
# www.sagemath.org/pdf/en/tutorial/SageTutorial.pdf
```

```
# 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; # Ausgabe ist immer exakt
```

```
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'>
<type 'sage.rings.rational.Rational'>
```

```
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
    7.38905609893065022723042746057500781318031557055184732408712782
    57379607905776338431248507912179477375316126547886612388460369
    33744783922133980777749001228956074107537023913309475506820865
```

```
infinity
```

```
    +Infinity
```

```
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-6.3/local/lib/python2.7/site-packages/sage/functions/log.py

Type: <class 'sage.functions.log.Function_log'>

Definition: log(*args, **kwds)

Docstring:

The natural logarithm of x. See $log?$ for more information about its behavior.

EXAMPLES:

```
sage: ln(e^2)
2
sage: ln(2)
log(2)
sage: ln(10)
log(10)
```

```
sage: ln(RDF(10))
2.30258509299
sage: ln(2.718)
0.999896315728952
sage: ln(2.0)
0.693147180559945
sage: ln(float(-1))
3.141592653589793j
sage: ln(complex(-1))
3.141592653589793j
```

The `hold` parameter can be used to prevent automatic evaluation:

```
sage: log(-1,hold=True)
log(-1)
sage: log(-1)
I*pi
sage: I.log(hold=True)
log(I)
sage: I.log(hold=True).simplify()
1/2*I*pi
```

TESTS:

```
sage: latex(x.log())
\log\left(x\right)
sage: latex(log(1/4))
\log\left(\frac{1}{4}\right)
sage: loads(dumps(ln(x)+1))
log(x) + 1
```

`conjugate(log(x))==log(conjugate(x))` unless on the branch cut which runs along the negative real axis.:

```
sage: conjugate(log(x))
conjugate(log(x))
sage: var('y', domain='positive')
y
sage: conjugate(log(y))
log(y)
sage: conjugate(log(y+I))
conjugate(log(y + I))
sage: conjugate(log(-1))
-I*pi
sage: log(conjugate(-1))
I*pi
```

Check if float arguments are handled properly.:

```
sage: from sage.functions.log import function_log as log
sage: log(float(5))
1.6094379124341003
```

```
sage: log(float(0))
-inf
sage: log(float(-1))
3.141592653589793j
sage: log(x).subs(x=float(-1))
3.141592653589793j
```

```
exp(x^2+log(x))
```
```
e^(x^2 + log(x))
```
```
_.simplify()
```
```
x*e^(x^2)
```
```
n(tan(3))
```
```
-0.142546543074278
```
```
abs(-2)
```
```
2
```
```
ceil(3.44);floor(3.44);round(3.44);
```
```
4
3
3
```
```
# Aussagen / Logik
```

```
true
```
```
True
```
```
type(true)
```
```
<type 'bool'>
```
```
is_even(2)
```
```
True
```
```
is_odd(2)
```
```
False
```
```
6%2==0
```
```
True
```
```
6%4==0
```
```
False
```
```
2<9
```
```
True
```
```
true|false
```
```
True
```
```
true&false
```

False

```
propcalc?
```

File: /home/sage/sage-6.3/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 a
explicitly show order of operation.

AUTHORS:

- Chris Gorecki (2006): initial version, propcalc, boolformula, logictable, logicparser, boolev
- Michael Greenberg – boolopt
- Paul Scurek (2013-08-05): updated docstring formatting
- Paul Scurek (2013-08-12): added `get_formulas()`, `consistent()`, `valid_consequenc`

EXAMPLES:

We can create boolean formulas in different ways:

```
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
```

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

```
sage: propcalc.formula("")
Traceback (most recent call last):
...
SyntaxError: malformed statement
sage: propcalc.formula("a&b~(c|(d)")
Traceback (most recent call last):
...
SyntaxError: malformed statement
```

```
sage: propcalc.formula("a&&b")
Traceback (most recent call last):
...
SyntaxError: malformed statement
sage: propcalc.formula("a&b a")
Traceback (most recent call last):
...
SyntaxError: malformed statement

It is also an error to not abide by the naming conventions.
sage: propcalc.formula("~a&9b")
Traceback (most recent call last):
...
NameError: invalid variable name 9b: identifiers must begin with a letter a
```

```
import sage.logic.propcalc as propcalc
```

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

```
    True
```

```
# Variablen
```

```
a=5
```

```
a
```

```
    5
```

```
a==5
```

```
    True
```

```
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
```

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

```
y=var('y');y # y ist nun leere Variable
```

```
    y
```

```
reset() # alle Variablen löschen
```

```
x;y;
```

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

```
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;
```

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

```
s.simplify() # zum Vereinfachen von Ausdrücken
```

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

```
s.simplify_trig() # vereinfach trigonometrische Ausdrücke
```

```
    1
```

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

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

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

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

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

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

```
# 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])
```

```
M1.difference(M2) # Differenz
```

```
    set([3, 5])
```

```
6 in M2
```

```
    True
```

```
9 in M2
```

```
    False
```

```
M1==M2
```

```
    False
```

```
# Vektoren und Matrizen
```

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

```
    <type
    'sage.modules.vector_integer_dense.Vector_integer_dense'>
```

```
a
```

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

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

```
a+b
```

```
    (5, 7, 9)
```

```
a.get(0) # zählt Eintrage ab 0 (=erster Eintrag)
```

```
    1
```

```
a[0]
```

```
    1
```

```
c=2.1
```

```
a*c;a*b; # komponentenweise Multiplikation; Skalarprodukt
```

```
(2.10000000000000, 4.20000000000000, 6.30000000000000)
32
```

```
d=a.outer_product(b)
```

```
d
```

```
[ 4  5  6]
[ 8 10 12]
[12 15 18]
```

```
type(d)
```

```
<type 'sage.matrix.matrix_integer_dense.Matrix_integer_dense'>
```

```
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.transpose()
```

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

```
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;M
```

```
[0 0 0 0]
[0 0 0 0]
```

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

```
N[:,3]=a;N
```

```
[0 0 0 1]
[0 0 0 2]
[0 0 0 3]
```

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

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

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

```
3
3
```

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

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

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

```
x
```

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

```
M\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'>
```

```
vec=vector(v);mengev=set(v);
```

```
type(vec);type(mengev);
```

```
    <type
    'sage.modules.vector_integer_dense.Vector_integer_dense'>
    <type 'set'>
```

```
v[0]
```

```
    2
```

```
v=v+[2]
```

```
v
```

```
    [2, 3, 5, 2]
```

```
v[0]=1
```

```
v
```

```
    [1, 3, 5, 2]
```

```
v=v+['hallo'];v
```

```
    [1, 3, 5, 2, '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]
```

```
L=[2^n for n in [1,2,3,1,2,3]]; L
```

```
    [2, 4, 8, 2, 4, 8]
```

```
M=set(L);M
```

```
    set([8, 2, 4])
```

```
L=[factor(n) for n in range(1,50) if is_odd(n)] #
Primfaktorzerlegung für gerade n
```

```
L
```

```
    [1, 3, 5, 7, 3^2, 11, 13, 3 * 5, 17, 19, 3 * 7, 23, 5^2, 3^3, 1
    31, 3 * 11, 5 * 7, 37, 3 * 13, 41, 43, 3^2 * 5, 47, 7^2]
```

```
# Rechenbeispiele
```

```
n,i=var('n,i')
```

```
sum(i,i,1,n) # Summe über i für i von 1 bis n
```

```
    1/2*n^2 + 1/2*n
```

```
sum(i^5, i, 1, n)
```

```
    1/6*n^6 + 1/2*n^5 + 5/12*n^4 - 1/12*n^2
```

```
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-6.3/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 g

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 dictionari empty dictionary). Likewise, if there's only a single solution, return a list containing

There are a few optional keywords if you are trying to solve a single equation. They may c

- `multiplicities` - bool (default: False); if True, return corresponding multiplicities. inequalities.
- `explicit_solutions` - bool (default: False); require that all roots be explicit rathe
- `to_poly_solve` - bool (default: False) or string; use Maxima's `to_poly_solver` p incompatible with `multiplicities=True` and is not used when solving inequalities trigonometric equations are lost).

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
sage: solutions=solve([x^2+y^2 == 1, y^2 == x^3 + x + 1], x, y, soluti
sage: for solution in solutions: print solution[x].n(digits=3), ",", s
-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 app

```
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 == (-
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

```
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/
sage: solve( [y^6 == y], y)==solve( y^6 == y, y)
True
```

Here we demonstrate very basic use of the optional keywords for a single expression to b

```
sage: ((x^2-1)^2).solve(x)
[x == -1, x == 1]
sage: ((x^2-1)^2).solve(x,multiplicities=True)
([x == -1, x == 1], [2, 2])
sage: solve(sin(x)==x,x)
[x == sin(x)]
sage: solve(sin(x)==x,x,explicit_solutions=True)
[]
sage: solve(abs(1-abs(1-x)) == 10, x)
[abs(abs(x - 1) - 1) == 10]
sage: solve(abs(1-abs(1-x)) == 10, x, to_poly_solve=True)
[x == -10, x == 12]
```

Note

For more details about solving a single equation, see the documentation for the single-e

```
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)
Traceback (most recent call last):
...
TypeError: 5 is not a valid variable.
```

If we ask for dictionaries containing the solutions, we get them:

```
sage: solve([x^2-1],x,solution_dict=True)
[{x: -1}, {x: 1}]
sage: solve([x^2-4*x+4],x,solution_dict=True)
[{x: 2}]
sage: res = solve([x^2 == y, y == 4],x,y,solution_dict=True)
sage: for soln in res: print "x: %s, y: %s"%(soln[x], soln[y])
x: 2, y: 4
x: -2, y: 4
```

If there is a parameter in the answer, that will show up as a new variable. In the following

```
sage: solve([x+y == 3, 2*x+2*y == 6],x,y)
[[x == -r1 + 3, y == r1]]
```

Especially with trigonometric functions, the dummy variable may be implicitly an integer

```
sage: solve([cos(x)*sin(x) == 1/2, x+y == 0],x,y)
[[x == 1/4*pi + pi*z80, y == -1/4*pi - pi*z80]]
```

Expressions which are not equations are assumed to be set equal to zero, as with x in th

```
sage: solve([x, y == 2],x,y)
[[x == 0, y == 2]]
```

If `True` appears in the list of equations it is ignored, and if `False` appears in the list then

```
sage: solve([3==3, 1.00000000000000*x^3 == 0], x)
[x == 0]
```

```
sage: solve([1.00000000000000*x^3 == 0], x)
[x == 0]
```

Here, the first equation evaluates to `False`, so there are no solutions:

```
sage: solve([1==3, 1.00000000000000*x^3 == 0], x)
[]
```

Completely symbolic solutions are supported:

```
sage: var('s,j,b,m,g')
(s, j, b, m, g)
sage: sys = [ m*(1-s) - b*s*j, b*s*j-g*j ];
sage: solve(sys,s,j)
[[s == 1, j == 0], [s == g/b, j == (b - g)*m/(b*g)]]
sage: solve(sys,(s,j))
[[s == 1, j == 0], [s == g/b, j == (b - g)*m/(b*g)]]
sage: solve(sys,[s,j])
[[s == 1, j == 0], [s == g/b, j == (b - g)*m/(b*g)]]
```

Inequalities can be also solved:

```
sage: solve(x^2>8,x)
[[x < -2*sqrt(2)], [x > 2*sqrt(2)]]
```

We use `use_grobner` in Maxima if no solution is obtained from Maxima's `to_poly_solv`

```
sage: x,y=var('x y'); c1(x,y)=(x-5)^2+y^2-16; c2(x,y)=(y-3)^2+x^2-9
sage: solve([c1(x,y),c2(x,y)],[x,y])
[[x == -9/68*sqrt(55) + 135/68, y == -15/68*sqrt(11)*sqrt(5) + 123/68]
```

TESTS:

```
sage: solve([sin(x)==x,y^2==x],x,y)
[sin(x) == x, y^2 == x]
sage: solve(0==1,x)
Traceback (most recent call last):
...
TypeError:  The first argument must be a symbolic expression or a list
```

Test if the empty list is returned, too, when (a list of) dictionaries (is) are requested (#85

```
sage: solve([SR(0)==1],x)
[]
sage: solve([SR(0)==1],x,solution_dict=True)
[]
sage: solve([x==1,x==-1],x)
[]
sage: solve([x==1,x==-1],x,solution_dict=True)
[]
sage: solve((x==1,x==-1),x,solution_dict=0)
[]
```

Relaxed form, suggested by Mike Hansen (#8553):

```
sage: solve([x^2-1],x,solution_dict=-1)
[{x: -1}, {x: 1}]
sage: solve([x^2-1],x,solution_dict=1)
[{x: -1}, {x: 1}]
sage: solve((x==1,x==-1),x,solution_dict=-1)
[]
```

```
sage: solve((x==1,x==-1),x,solution_dict=1)
[]
```

This inequality holds for any real x (trac #8078):

```
sage: solve(x^4+2>0,x)
[x < +Infinity]
```

Test for user friendly input handling trac ticket #13645:

```
sage: poly.<a,b> = PolynomialRing(RR)
sage: solve([a+b+a*b == 1], a)
Traceback (most recent call last):
...
TypeError: The first argument to solve() should be a symbolic expressi
sage: solve([a, b], (1, a))
Traceback (most recent call last):
...
TypeError: 1 is not a valid variable.
sage: solve([x == 1], (1, a))
Traceback (most recent call last):
...
TypeError: (1, a) are not valid variables.
```

Test that the original version of a system in the French Sage book now works (trac ticket

```
sage: var('y,z')
(y, z)
sage: solve([x^2*y*z==18,x*y^3*z==24,x*y*z^4==6],x,y,z)
[[x == 3, y == 2, z == 1], [x == (1.33721506733 - 2.68548987407*I), y
```

```
sage: x, y = var('x, y');
sage: solve([x+y==6, x-y==4], x, y);
```

```
    [[x == 5, y == 1]]
```

```
sudoku?
```

File: /home/sage/sage-6.3/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 repl

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
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 e

```
sage: sudoku('.4..32....14..3.')
Traceback (most recent call last):
...
ValueError: sudoku function expects puzzle to be a matrix, perhaps use
```

```
sage: A = matrix(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]
```

```
permutations?
```

File: /home/sage/sage-6.3/local/lib/python2.7/site-packages/sage/combinat/combinat.py

Type: <type 'function'>

Definition: permutations(mset)

Docstring:

This is deprecated in trac ticket #14772. Use `Permutations` instead. To get the same output as $permutations(mset)$, use `Permutations(mset).list()`.

A permutation is represented by a list that contains exactly the same elements as mset, but possibly in different order. If mset is a proper set there are $|mset|!$ such permutations. Otherwise if the first elements appears k_1 times, the second element appears k_2 times and so on, the number of permutations is $|mset|!/(k_1!k_2!\ldots)$, which is sometimes called a multinomial coefficient.

`permutations` returns the list of all permutations of a multiset.

EXAMPLES:

```
sage: mset = [1,1,2,2,2]
sage: permutations(mset)
doctest:...: DeprecationWarning: Use the Permutations object instead.
See http://trac.sagemath.org/14772 for details.
[[1, 1, 2, 2, 2],
 [1, 2, 1, 2, 2],
 [1, 2, 2, 1, 2],
 [1, 2, 2, 2, 1],
 [2, 1, 1, 2, 2],
 [2, 1, 2, 1, 2],
 [2, 1, 2, 2, 1],
 [2, 2, 1, 1, 2],
 [2, 2, 1, 2, 1],
 [2, 2, 2, 1, 1]]
sage: MS = MatrixSpace(GF(2),2,2)
sage: A = MS([1,0,1,1])
sage: rows = A.rows()
sage: rows[0].set_immutable()
sage: rows[1].set_immutable()
sage: permutations(rows)
[[(1, 0), (1, 1)], [(1, 1), (1, 0)]]
```

```
L=[2^n for n in range(1,4)];L
```

```
[2, 4, 8]
```

```
LL=Permutations(L)
```

```
LL.list()
```

```
[[2, 4, 8], [2, 8, 4], [4, 2, 8], [4, 8, 2], [8, 2, 4], [8, 4,
```

```
LL[3]
```

```
[4, 8, 2]
```

```
```