Sage Reference Manual: Interpreter Interfaces

Release 6.3

The Sage Development Team

August 11, 2014

CONTENTS

1 Common Interface Functionality through Pexpect **3**

2 Interface to Axiom **7**

3 The Elliptic Curve Factorization Method **13**

4 Interface to 4ti2 (http://www.4ti2.de) **19**

5 Interface to GAP **25**
- 5.1 First Examples . . . 25
- 5.2 GAP and Singular . . . 25
- 5.3 Saving and loading objects . . . 26
- 5.4 Long Input . . . 26
- 5.5 Changing which GAP is used . . . 27

6 Interface to GAP3 **37**
- 6.1 Obtaining GAP3 . . . 37
- 6.2 Changing which GAP3 is used . . . 37
- 6.3 Functionality and Examples . . . 38
- 6.4 Common Pitfalls . . . 39
- 6.5 Examples . . . 39

7 Interface to the GP calculator of PARI/GP **47**

8 Interface to the Gnuplot interpreter **57**

9 Interface to KASH **59**
- 9.1 Issues . . . 59
- 9.2 Tutorial . . . 59
- 9.3 Long Input . . . 65

10 Interface to Magma **67**
- 10.1 Parameters . . . 67
- 10.2 Multiple Return Values . . . 67
- 10.3 Long Input . . . 68
- 10.4 Garbage Collection . . . 68
- 10.5 Other Examples . . . 69

11 Interface to Maple **85**
- 11.1 Tutorial . . . 85

12 Interface to MATLAB 93
12.1 Tutorial . . . 93

13 Pexpect interface to Maxima 97
13.1 Tutorial . . . 98
13.2 Examples involving matrices . . . 100
13.3 Laplace Transforms . . . 100
13.4 Continued Fractions . . . 101
13.5 Special examples . . . 101
13.6 Miscellaneous . . . 102
13.7 Interactivity . . . 102
13.8 Latex Output . . . 102
13.9 Long Input . . . 103

14 Library interface to Maxima 107

15 Interface to Mathematica 121
15.1 Tutorial . . . 122
15.2 Long Input . . . 125
15.3 Loading and saving . . . 125
15.4 Complicated translations . . . 125

16 Interface to mwrank 129

17 Interface to GNU Octave 133
17.1 Computation of Special Functions . . . 133
17.2 Tutorial . . . 134

18 Interface to R 139

19 Interface to Sage 151

20 Interface to Singular 157
20.1 Introduction . . . 157
20.2 Tutorial . . . 157
20.3 Computing the Genus . . . 160
20.4 An Important Concept . . . 160
20.5 Long Input . . . 161

21 The Tachyon Ray Tracer 175

22 Interface for extracting data and generating images from Jmol readable files. 177

23 Indices and Tables 179

Sage provides a unified interface to the best computational software. This is accomplished using both C-libraries (see C/C++ Library Interfaces) and interpreter interfaces, which are implemented using pseudo-tty's, system files, etc. This chapter is about these interpreter interfaces.

Note: Each interface requires that the corresponding software is installed on your computer. Sage includes GAP, PARI, Singular, and Maxima, but does not include Octave (very easy to install), MAGMA (non-free), Maple (non-free), or Mathematica (non-free).

There is overhead associated with each call to one of these systems. For example, computing `2+2` thousands of times using the GAP interface will be slower than doing it directly in Sage. In contrast, the C-library interfaces of C/C++ Library Interfaces incur less overhead.

In addition to the commands described for each of the interfaces below, you can also type e.g., `%gap`, `%magma`, etc., to directly interact with a given interface in its state. Alternatively, if `X` is an interface object, typing `X.interact()` allows you to interact with it. This is completely different than `X.console()` which starts a complete new copy of whatever program `X` interacts with. Note that the input for `X.interact()` is handled by Sage, so the history buffer is the same as for Sage, tab completion is as for Sage (unfortunately!), and input that spans multiple lines must be indicated using a backslash at the end of each line. You can pull data into an interactive session with `X` using `sage(expression)`.

The console and interact methods of an interface do very different things. For example, using gap as an example:

1. `gap.console()`: You are completely using another program, e.g., gap/magma/gp Here Sage is serving as nothing more than a convenient program launcher, similar to bash.
2. `gap.interact()`: This is a convenient way to interact with a running gap instance that may be "full of" Sage objects. You can import Sage objects into this gap (even from the interactive interface), etc.

The console function is very useful on occasion, since you get the exact actual program available (especially useful for tab completion and testing to make sure nothing funny is going on).

CHAPTER

ONE

COMMON INTERFACE FUNCTIONALITY THROUGH PEXPECT

See the examples in the other sections for how to use specific interfaces. The interface classes all derive from the generic interface that is described in this section.

AUTHORS:

- William Stein (2005): initial version
- William Stein (2006-03-01): got rid of infinite loop on startup if client system missing
- Felix Lawrence (2009-08-21): edited ._sage_() to support lists and float exponents in foreign notation.
- Simon King (2010-09-25): Expect._local_tmpfile() depends on Expect.pid() and is cached; Expect.quit() clears that cache, which is important for forking.
- Jean-Pierre Flori (2010,2011): Split non Pexpect stuff into a parent class.
- Simon King (2010-11-23): Ensure that the interface is started again after a crash, when a command is executed in _eval_line. Allow synchronisation of the GAP interface.

class `sage.interfaces.expect.`**`Expect`**(*name*, *prompt*, *command=None*, *server=None*, *server_tmpdir=None*, *ulimit=None*, *maxread=100000*, *script_subdirectory=''*, *restart_on_ctrlc=False*, *verbose_start=False*, *init_code=[]*, *max_startup_time=None*, *logfile=None*, *eval_using_file_cutoff=0*, *do_cleaner=True*, *remote_cleaner=False*, *path=None*, *terminal_echo=True*)

Bases: `sage.interfaces.interface.Interface`

Expect interface object.

`clear_prompts`()

x.__init__(...) initializes x; see help(type(x)) for signature

`eval`(*code*, *strip=True*, *synchronize=False*, *locals=None*, *allow_use_file=True*, *split_lines='nofile'*, ***kwds*)

INPUT:

- `code` – text to evaluate
- **`strip` – bool; whether to strip output prompts,** etc. (ignored in the base class).
- **`locals` – None (ignored); this is used for compatibility** with the Sage notebook's generic system interface.

- **allow_use_file – bool (default: True); if True and code exceeds an** interface-specific threshold then `code` will be communicated via a temporary file rather that the character-based interface. If False then the code will be communicated via the character interface.
- **split_lines – Tri-state (default: "nofile"); if "nofile" then code is sent line by line** unless it gets communicated via a temporary file. If True then `code` is sent line by line, but some lines individually might be sent via temporary file. Depending on the interface, this may transform grammatical `code` into ungrammatical input. If False, then the whole block of code is evaluated all at once.
- ****kwds – All other arguments are passed onto the _eval_line** method. An often useful example is reformat=False.

expect()
x.__init__(...) initializes x; see help(type(x)) for signature

interrupt(*tries=20*, *timeout=0.3*, *quit_on_fail=True*)
x.__init__(...) initializes x; see help(type(x)) for signature

is_local()
x.__init__(...) initializes x; see help(type(x)) for signature

is_remote()
x.__init__(...) initializes x; see help(type(x)) for signature

is_running()
Return True if self is currently running.

path()
x.__init__(...) initializes x; see help(type(x)) for signature

pid()
Return the PID of the underlying sub-process.

REMARK:

If the interface terminates unexpectedly, the original PID will still be used. But if it was terminated using `quit()`, a new sub-process with a new PID is automatically started.

EXAMPLE:

```
sage: pid = gap.pid()
sage: gap.eval('quit;')
''
sage: pid == gap.pid()
True
sage: gap.quit()
sage: pid == gap.pid()
False
```

quit(*verbose=False*, *timeout=0.25*)
EXAMPLES:

```
sage: a = maxima('y')
sage: maxima.quit()
sage: a._check_valid()
Traceback (most recent call last):
...
ValueError: The maxima session in which this object was defined is no longer running.
```

read(*filename*)
EXAMPLES:

```
sage: filename = tmp_filename()
sage: f = open(filename, 'w')
sage: f.write('x = 2\n')
sage: f.close()
sage: octave.read(filename)  # optional - octave
sage: octave.get('x')        #optional
' 2'
sage: import os
sage: os.unlink(filename)
```

user_dir()
x.__init__(...) initializes x; see help(type(x)) for signature

class sage.interfaces.expect.**ExpectElement**(*parent*, *value*, *is_name=False*, *name=None*)
Bases: sage.interfaces.interface.InterfaceElement

Expect element.

class sage.interfaces.expect.**ExpectFunction**(*parent*, *name*)
Bases: sage.interfaces.interface.InterfaceFunction

Expect function.

class sage.interfaces.expect.**FunctionElement**(*obj*, *name*)
Bases: sage.interfaces.interface.InterfaceFunctionElement

Expect function element.

class sage.interfaces.expect.**StdOutContext**(*interface*, *silent=False*, *stdout=None*)
A context in which all communation between Sage and a subprocess interfaced via pexpect is printed to stdout.

sage.interfaces.expect.**console**(*cmd*)
x.__init__(...) initializes x; see help(type(x)) for signature

class sage.interfaces.expect.**gc_disabled**
Bases: object

This is a "with" statement context manager. Garbage collection is disabled within its scope. Nested usage is properly handled.

EXAMPLES:

```
sage: import gc
sage: from sage.interfaces.expect import gc_disabled
sage: gc.isenabled()
True
sage: with gc_disabled():
...       print gc.isenabled()
...       with gc_disabled():
...           print gc.isenabled()
...       print gc.isenabled()
False
False
False
sage: gc.isenabled()
True
```

sage.interfaces.expect.**is_ExpectElement**(*x*)
x.__init__(...) initializes x; see help(type(x)) for signature

CHAPTER

TWO

INTERFACE TO AXIOM

TODO:

- Evaluation using a file is not done. Any input line with more than a few thousand characters would hang the system, so currently it automatically raises an exception.
- All completions of a given command.
- Interactive help.

Axiom is a free GPL-compatible (modified BSD license) general purpose computer algebra system whose development started in 1973 at IBM. It contains symbolic manipulation algorithms, as well as implementations of special functions, including elliptic functions and generalized hypergeometric functions. Moreover, Axiom has implementations of many functions relating to the invariant theory of the symmetric group S_n. For many links to Axiom documentation see http://wiki.axiom-developer.org.

AUTHORS:

- Bill Page (2006-10): Created this (based on Maxima interface)

  Note: Bill Page put a huge amount of effort into the Sage Axiom interface over several days during the Sage Days 2 coding sprint. This is contribution is greatly appreciated.

- William Stein (2006-10): misc touchup.
- Bill Page (2007-08): Minor modifications to support axiom4sage-0.3

Note: The axiom4sage-0.3.spkg is based on an experimental version of the FriCAS fork of the Axiom project by Waldek Hebisch that uses pre-compiled cached Lisp code to build Axiom very quickly with clisp.

If the string "error" (case insensitive) occurs in the output of anything from axiom, a RuntimeError exception is raised.

EXAMPLES: We evaluate a very simple expression in axiom.

```
sage: axiom('3 * 5')                     #optional - axiom
15
sage: a = axiom(3) * axiom(5); a         #optional - axiom
15
```

The type of a is AxiomElement, i.e., an element of the axiom interpreter.

```
sage: type(a)                            #optional - axiom
<class 'sage.interfaces.axiom.AxiomElement'>
sage: parent(a)                          #optional - axiom
Axiom
```

The underlying Axiom type of a is also available, via the type method:

```
sage: a.type()                          #optional - axiom
PositiveInteger
```

We factor $x^5 - y^5$ in Axiom in several different ways. The first way yields a Axiom object.

```
sage: F = axiom.factor('x^5 - y^5'); F       #optional - axiom
           4       3     2 2     3     4
- (y - x)(y  + x y  + x y  + x y + x )
sage: type(F)                                #optional - axiom
<class 'sage.interfaces.axiom.AxiomElement'>
sage: F.type()                               #optional - axiom
Factored Polynomial Integer
```

Note that Axiom objects are normally displayed using "ASCII art".

```
sage: a = axiom(2/3); a          #optional - axiom
  2
  -
  3
sage: a = axiom('x^2 + 3/7'); a      #optional - axiom
   2   3
  x  + -
       7
```

The `axiom.eval` command evaluates an expression in axiom and returns the result as a string. This is exact as if we typed in the given line of code to axiom; the return value is what Axiom would print out.

```
sage: print axiom.eval('factor(x^5 - y^5)')   #optional - axiom
           4       3     2 2     3     4
- (y - x)(y  + x y  + x y  + x y + x )
Type: Factored Polynomial Integer
```

We can create the polynomial f as a Axiom polynomial, then call the factor method on it. Notice that the notation `f.factor()` is consistent with how the rest of Sage works.

```
sage: f = axiom('x^5 - y^5')                   #optional - axiom
sage: f^2                                      #optional - axiom
   10      5 5     10
  y   - 2x y  + x
sage: f.factor()                               #optional - axiom
           4       3     2 2     3     4
- (y - x)(y  + x y  + x y  + x y + x )
```

Control-C interruption works well with the axiom interface, because of the excellent implementation of axiom. For example, try the following sum but with a much bigger range, and hit control-C.

```
sage:  f = axiom('(x^5 - y^5)^10000')       # not tested
Interrupting Axiom...
...
<type 'exceptions.TypeError'>: Ctrl-c pressed while running Axiom
```

```
sage: axiom('1/100 + 1/101')                   #optional - axiom
   201
  -----
  10100
sage: a = axiom('(1 + sqrt(2))^5'); a          #optional - axiom
```

```
     +-+
  29\|2  + 41
```

TESTS: We check to make sure the subst method works with keyword arguments.

```
sage: a = axiom(x+2); a  #optional - axiom
x + 2
sage: a.subst(x=3)       #optional - axiom
5
```

We verify that Axiom floating point numbers can be converted to Python floats.

```
sage: float(axiom(2))      #optional - axiom
2.0
```

class `sage.interfaces.axiom.`**`Axiom`**(*name='axiom', command='axiom -nox -noclef', script_subdirectory=None, logfile=None, server=None, server_tmpdir=None, init_code=[')lisp (si::readline-off)']*)

Bases: `sage.interfaces.axiom.PanAxiom`

Create an instance of the Axiom interpreter.

TESTS:

```
sage: axiom == loads(dumps(axiom))
True
```

`console`()

Spawn a new Axiom command-line session.

EXAMPLES:

```
sage: axiom.console() #not tested
                        AXIOM Computer Algebra System
                        Version: Axiom (January 2009)
               Timestamp: Sunday January 25, 2009 at 07:08:54
-----------------------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave AXIOM and return to shell.
-----------------------------------------------------------------------------
```

class `sage.interfaces.axiom.`**`AxiomElement`**(*parent, value, is_name=False, name=None*)

Bases: `sage.interfaces.axiom.PanAxiomElement`

class `sage.interfaces.axiom.`**`AxiomExpectFunction`**(*parent, name*)

Bases: `sage.interfaces.axiom.PanAxiomExpectFunction`

TESTS:

```
sage: axiom.upperCase_q
upperCase?
sage: axiom.upperCase_e
upperCase!
```

class `sage.interfaces.axiom.`**`AxiomFunctionElement`**(*object, name*)

Bases: `sage.interfaces.axiom.PanAxiomFunctionElement`

TESTS:

```
sage: a = axiom('"Hello"') #optional - axiom
sage: a.upperCase_q        #optional - axiom
upperCase?
sage: a.upperCase_e        #optional - axiom
upperCase!
sage: a.upperCase_e()      #optional - axiom
"HELLO"
```

class `sage.interfaces.axiom.`**`PanAxiom`**(*name='axiom', command='axiom -nox -noclef', script_subdirectory=None, logfile=None, server=None, server_tmpdir=None, init_code=[')lisp (si::readline-off)']*)

Bases: `sage.interfaces.expect.Expect`

Interface to a PanAxiom interpreter.

`get`(*var*)

Get the string value of the Axiom variable var.

EXAMPLES:

```
sage: axiom.set('xx', '2')    #optional - axiom
sage: axiom.get('xx')         #optional - axiom
'2'
sage: a = axiom('(1 + sqrt(2))^5') #optional - axiom
sage: axiom.get(a.name())          #optional - axiom
'      +-+\r\r\n  29\\|2  + 41'
```

`set`(*var, value*)

Set the variable var to the given value.

EXAMPLES:

```
sage: axiom.set('xx', '2')    #optional - axiom
sage: axiom.get('xx')         #optional - axiom
'2'

sage: fricas.set('xx', '2')    #optional - fricas
sage: fricas.get('xx')         #optional - fricas
'2'
```

`trait_names`(*verbose=True, use_disk_cache=True*)

Returns a list of all the commands defined in Axiom and optionally (per default) store them to disk.

EXAMPLES:

```
sage: c = axiom.trait_names(use_disk_cache=False, verbose=False) #optional - axiom
sage: len(c) > 100  #optional - axiom
True
sage: 'factor' in c  #optional - axiom
True
sage: '**' in c     #optional - axiom
False
sage: 'upperCase?' in c  #optional - axiom
False
sage: 'upperCase_q' in c #optional - axiom
True
sage: 'upperCase_e' in c #optional - axiom
True
```

class `sage.interfaces.axiom.`**`PanAxiomElement`**(*parent*, *value*, *is_name=False*, *name=None*)
Bases: `sage.interfaces.expect.ExpectElement`

`as_type`(*type*)
Returns self as type.

EXAMPLES:

```
sage: a = axiom(1.2); a            #optional - axiom
1.2
sage: a.as_type(axiom.DoubleFloat) #optional - axiom
1.2
sage: _.type()                     #optional - axiom
DoubleFloat

sage: a = fricas(1.2); a            #optional - fricas
1.2
sage: a.as_type(fricas.DoubleFloat) #optional - fricas
1.2
sage: _.type()                      #optional - fricas
DoubleFloat
```

`comma`(**args*)
Returns a Axiom tuple from self and args.

EXAMPLES:

```
sage: two = axiom(2)  #optional - axiom
sage: two.comma(3)    #optional - axiom
[2,3]
sage: two.comma(3,4)  #optional - axiom
[2,3,4]
sage: _.type()        #optional - axiom
Tuple PositiveInteger

sage: two = fricas(2)  #optional - fricas
sage: two.comma(3)     #optional - fricas
[2,3]
sage: two.comma(3,4)   #optional - fricas
[2,3,4]
sage: _.type()         #optional - fricas
Tuple(PositiveInteger)
```

`type`()
Returns the type of an AxiomElement.

EXAMPLES:

```
sage: axiom(x+2).type()  #optional - axiom
Polynomial Integer
```

`unparsed_input_form`()
Get the linear string representation of this object, if possible (often it isn't).

EXAMPLES:

```
sage: a = axiom(x^2+1); a     #optional - axiom
   2
  x  + 1
sage: a.unparsed_input_form() #optional - axiom
'x*x+1'
```

```
sage: a = fricas(x^2+1)        #optional - fricas
sage: a.unparsed_input_form() #optional - fricas
'x^2+1'
```

class sage.interfaces.axiom.**PanAxiomExpectFunction**(*parent*, *name*)

Bases: sage.interfaces.expect.ExpectFunction

TESTS:

```
sage: axiom.upperCase_q
upperCase?
sage: axiom.upperCase_e
upperCase!
```

class sage.interfaces.axiom.**PanAxiomFunctionElement**(*object*, *name*)

Bases: sage.interfaces.expect.FunctionElement

TESTS:

```
sage: a = axiom('"Hello"') #optional - axiom
sage: a.upperCase_q        #optional - axiom
upperCase?
sage: a.upperCase_e        #optional - axiom
upperCase!
sage: a.upperCase_e()      #optional - axiom
"HELLO"
```

sage.interfaces.axiom.**axiom_console**()

Spawn a new Axiom command-line session.

EXAMPLES:

```
sage: axiom_console() #not tested
                        AXIOM Computer Algebra System
                        Version: Axiom (January 2009)
              Timestamp: Sunday January 25, 2009 at 07:08:54
-----------------------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave AXIOM and return to shell.
-----------------------------------------------------------------------------
```

sage.interfaces.axiom.**is_AxiomElement**(*x*)

Returns True of x is of type AxiomElement.

EXAMPLES:

```
sage: from sage.interfaces.axiom import is_AxiomElement
sage: is_AxiomElement(axiom(2)) #optional - axiom
True
sage: is_AxiomElement(2)
False
```

sage.interfaces.axiom.**reduce_load_Axiom**()

Returns the Axiom interface object defined in sage.interfaces.axiom.

EXAMPLES:

```
sage: from sage.interfaces.axiom import reduce_load_Axiom
sage: reduce_load_Axiom()
Axiom
```

CHAPTER

THREE

THE ELLIPTIC CURVE FACTORIZATION METHOD

The elliptic curve factorization method (ECM) is the fastest way to factor a **known composite** integer if one of the factors is relatively small (up to approximately 80 bits / 25 decimal digits). To factor an arbitrary integer it must be combined with a primality test. The `ECM.factor()` method is an example for how to combine ECM with a primality test to compute the prime factorization of integers.

Sage includes GMP-ECM, which is a highly optimized implementation of Lenstra's elliptic curve factorization method. See http://ecm.gforge.inria.fr for more about GMP-ECM.

AUTHORS:

These people wrote GMP-ECM: Pierrick Gaudry, Jim Fougeron, Laurent Fousse, Alexander Kruppa, Dave Newman, Paul Zimmermann

BUGS:

Output from ecm is non-deterministic. Doctests should set the random seed, but currently there is no facility to do so.

class `sage.interfaces.ecm.`**`ECM`**(*B1=10*, *B2=None*, ***kwds*)

Bases: `sage.structure.sage_object.SageObject`

Create an interface to the GMP-ECM elliptic curve method factorization program.

See http://ecm.gforge.inria.fr

INPUT:

- `B1` – integer. Stage 1 bound
- `B2` – integer. Stage 2 bound (or interval B2min-B2max)

In addition the following keyword arguments can be used:

- `x0` – integer x. use x as initial point
- `sigma` – integer s. Use s as curve generator [ecm]
- `A` – integer a. Use a as curve parameter [ecm]
- `k` – integer n. Perform $>= n$ steps in stage 2
- `power` – integer n. Use x^n for Brent-Suyama's extension
- `dickson` – integer n. Use n-th Dickson's polynomial for Brent-Suyama's extension
- `c` – integer n. Perform n runs for each input
- `pm1` – boolean. perform P-1 instead of ECM

- pp1 – boolean. perform P+1 instead of ECM
- q – boolean. quiet mode
- v – boolean. verbose mode
- timestamp – boolean. print a time stamp with each number
- mpzmod – boolean. use GMP's mpz_mod for mod reduction
- modmuln – boolean. use Montgomery's MODMULN for mod reduction
- redc – boolean. use Montgomery's REDC for mod reduction
- nobase2 – boolean. Disable special base-2 code
- base2 – integer n. Force base 2 mode with 2^n+1 (n>0) or 2^n-1 (n<0)
- save – string filename. Save residues at end of stage 1 to file
- savea – string filename. Like -save, appends to existing files
- resume – string filename. Resume residues from file, reads from stdin if file is "-"
- primetest – boolean. Perform a primality test on input
- treefile – string. Store product tree of F in files f.0 f.1 ...
- i – integer. increment B1 by this constant on each run
- I – integer f. auto-calculated increment for B1 multiplied by f scale factor.
- inp – string. Use file as input (instead of redirecting stdin)
- b – boolean. Use breadth-first mode of file processing
- d – boolean. Use depth-first mode of file processing (default)
- one – boolean. Stop processing a candidate if a factor is found (looping mode)
- n – boolean. Run ecm in 'nice' mode (below normal priority)
- nn – boolean. Run ecm in 'very nice' mode (idle priority)
- t – integer n. Trial divide candidates before P-1, P+1 or ECM up to n.
- ve – integer n. Verbosely show short ($< n$ character) expressions on each loop
- cofdec – boolean. Force cofactor output in decimal (even if expressions are used)
- B2scale – integer. Multiplies the default B2 value
- go – integer. Preload with group order val, which can be a simple expression, or can use N as a placeholder for the number being factored.
- prp – string. use shell command cmd to do large primality tests
- prplen – integer. only candidates longer than this number of digits are 'large'
- prpval – integer. value>=0 which indicates the prp command foundnumber to be PRP.
- prptmp – file. outputs n value to temp file prior to running (NB. gets deleted)
- prplog – file. otherwise get PRP results from this file (NB. gets deleted)
- prpyes – string. Literal string found in prplog file when number is PRP
- prpno – string. Literal string found in prplog file when number is composite

factor (*n*, *factor_digits=None*, *B1=2000*, *proof=False*, ***kwds*)

Return a probable prime factorization of n.

Combines GMP-ECM with a primality test, see `is_prime()`. The primality test is provable or probabilistic depending on the $proof$ flag.

Moreover, for small n PARI is used directly.

> **Warning:** There is no mathematical guarantee that the factors returned are actually prime if `proof=False` (default). It is extremely likely, though. Currently, there are no known examples where this fails.

INPUT:

- `n` – a positive integer
- `factor_digits` – integer or `None` (default). Optional guess at how many digits are in the smallest factor.
- `B1` – initial lower bound, defaults to 2000 (15 digit factors). Used if `factor_digits` is not specified.
- `proof` – boolean (default: `False`). Whether to prove that the factors are prime.
- `kwds` – keyword arguments to pass to ecm-gmp. See help for `ECM` for more details.

OUTPUT:

A list of integers whose product is n.

Note: Trial division should typically be performed, but this is not implemented (yet) in this method.

If you suspect that n is the product of two similarly-sized primes, other methods (such as a quadratic sieve – use the qsieve command) will usually be faster.

The best known algorithm for factoring in the case where all factors are large is the general number field sieve. This is not implemented in Sage; You probably want to use a cluster for problems of this size.

EXAMPLES:

```
sage: ecm.factor(602400691612422154516282778947806249229526581)
[45949729863572179, 13109994191499930367061460439]
sage: ecm.factor((2^197 + 1)/3)  # long time
[197002597249, 1348959352853811313, 251951573867253012259144010843]
sage: ecm.factor(179427217^13) == [179427217] * 13
True
```

find_factor (*n*, *factor_digits=None*, *B1=2000*, ***kwds*)

Return a factor of n.

See also `factor()` if you want a prime factorization of n.

INPUT:

- `n` – a positive integer,
- `factor_digits` – integer or `None` (default). Decimal digits estimate of the wanted factor.
- `B1` – integer. Stage 1 bound (default 2000). This is used as bound if `factor_digits` is not specified.
- `kwds` – optional keyword parameters.

OUTPUT:

List of integers whose product is n. For certain lengths of the factor, this is the best algorithm to find a factor.

EXAMPLES:

```
sage: f = ECM()
sage: n = 508021860739623467191080372196682785441177798407961
sage: f.find_factor(n)
[79792266297612017, 6366805760909027985741435139224233]
```

Note that the input number can't have more than 4095 digits:

```
sage: f=2^2^14+1
sage: ecm.find_factor(f)
Traceback (most recent call last):
...
ValueError: n must have at most 4095 digits
```

get_last_params()

Return the parameters (including the curve) of the last ecm run.

In the case that the number was factored successfully, this will return the parameters that yielded the factorization.

OUTPUT:

A dictionary containing the parameters for the most recent factorization.

EXAMPLES:

```
sage: ecm.factor((2^197 + 1)/3)              # long time
[197002597249, 1348959352853811313, 251951573867253012259144010843]
sage: ecm.get_last_params()                  # random output
{'poly': 'x^1', 'sigma': '1785694449', 'B1': '8885', 'B2': '1002846'}
```

interact()

Interactively interact with the ECM program.

EXAMPLES:

```
sage: ecm.interact()     # not tested
```

one_curve(*n*, *factor_digits=None*, *B1=2000*, *algorithm='ECM'*, ***kwds*)

Run one single ECM (or P-1/P+1) curve on input n.

Note that trying a single curve is not particularly useful by itself. One typically needs to run over thousands of trial curves to factor n.

INPUT:

- `n` – a positive integer
- `factor_digits` – integer. Decimal digits estimate of the wanted factor.
- `B1` – integer. Stage 1 bound (default 2000)
- `algorithm` – either "ECM" (default), "P-1" or "P+1"

OUTPUT:

a list `[p, q]` where p and q are integers and n = p * q. If no factor was found, then p = 1 and q = n.

Warning: Neither p nor q in the output is guaranteed to be prime.

EXAMPLES:

```
sage: f = ECM()
sage: n = 508021860739623467191080372196682785441177798407961
sage: f.one_curve(n, B1=10000, sigma=11)
[1, 508021860739623467191080372196682785441177798407961]
sage: f.one_curve(n, B1=10000, sigma=1022170541)
[79792266297612017, 6366805760909027985741435139224233]
sage: n = 432132887883903108009802143314445113500016816977037257
sage: f.one_curve(n, B1=500000, algorithm="P-1")
[67872792749091946529, 6366805760909027985741435139224233]
sage: n = 2088352670731726262548647919416588631875815083
sage: f.one_curve(n, B1=2000, algorithm="P+1", x0=5)
[328006342451, 6366805760909027985741435139224233]
```

recommended_B1 (*factor_digits*)

Return recommended B1 setting.

INPUT:

- `factor_digits` – integer. Number of digits.

OUTPUT:

Integer. Recommended settings from http://www.mersennewiki.org/index.php/Elliptic_Curve_Method

EXAMPLES:

```
sage: ecm.recommended_B1(33)
1000000
```

time (*n*, *factor_digits*, *verbose=False*)

Print a runtime estimate.

BUGS:

This method should really return something and not just print stuff on the screen.

INPUT:

- `n` – a positive integer
- `factor_digits` – the (estimated) number of digits of the smallest factor

OUTPUT:

An approximation for the amount of time it will take to find a factor of size factor_digits in a single process on the current computer. This estimate is provided by GMP-ECM's verbose option on a single run of a curve.

EXAMPLES:

```
sage: n = next_prime(11^23)*next_prime(11^37)
sage: ecm.time(n, 35)                    # random output
Expected curves: 910, Expected time: 23.95m

sage: ecm.time(n, 30, verbose=True)      # random output
GMP-ECM 6.4.4 [configured with MPIR 2.6.0, --enable-asm-redc] [ECM]
Running on localhost.localdomain
Input number is 304481639541418099574459496544854621998616257489887231115912293 (63 digits)
Using MODMULN [mulredc:0, sqrredc:0]
```

```
Using B1=250000, B2=128992510, polynomial Dickson(3), sigma=3244548117
dF=2048, k=3, d=19110, d2=11, i0=3
Expected number of curves to find a factor of n digits:
35  40  45  50  55  60  65  70  75  80
4911  70940  1226976  2.5e+07  5.8e+08  1.6e+10  2.7e+13  4e+18  5.4e+23  Inf
Step 1 took 230ms
Using 10 small primes for NTT
Estimated memory usage: 4040K
Initializing tables of differences for F took 0ms
Computing roots of F took 9ms
Building F from its roots took 16ms
Computing 1/F took 9ms
Initializing table of differences for G took 0ms
Computing roots of G took 8ms
Building G from its roots took 16ms
Computing roots of G took 7ms
Building G from its roots took 16ms
Computing G * H took 6ms
Reducing  G * H mod F took 5ms
Computing roots of G took 7ms
Building G from its roots took 17ms
Computing G * H took 5ms
Reducing  G * H mod F took 5ms
Computing polyeval(F,G) took 34ms
Computing product of all F(g_i) took 0ms
Step 2 took 164ms
Expected time to find a factor of n digits:
35  40  45  50  55  60  65  70  75  80
32.25m  7.76h  5.60d  114.21d  7.27y  196.42y  337811y  5e+10y  7e+15y  Inf

Expected curves: 4911, Expected time: 32.25m
```

CHAPTER

FOUR

INTERFACE TO 4TI2 (HTTP://WWW.4TI2.DE)

You must have the 4ti2 Sage package installed on your computer for this interface to work.

AUTHORS:

- Mike Hansen (2009): Initial version.
- Bjarke Hammersholt Roune (2009-06-26): Added Groebner, made code usable as part of the Sage library and added documentation and some doctests.
- Marshall Hampton (2011): Minor fixes to documentation.

class `sage.interfaces.four_ti_2.`**`FourTi2`**(*directory=None*)

Bases: `object`

This object defines an interface to the program 4ti2. Each command 4ti2 has is exposed as one method.

`call`(*command*, *project*, *verbose=True*)

Run the 4ti2 program `command` on the project named `project` in the directory `directory()`.

INPUT:

- command - The 4ti2 program to run.
- project - The file name of the project to run on.
- verbose - Display the output of 4ti2 if `True`.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.write_matrix([[6,10,15]], "test_file")
sage: four_ti_2.call("groebner", "test_file", False) # optional - 4ti2
sage: four_ti_2.read_matrix("test_file.gro") # optional - 4ti2
[-5  0  2]
[-5  3  0]
```

`circuits`(*mat=None*, *project=None*)

Run the 4ti2 program `circuits` on the parameters. See `http://www.4ti2.de/` for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.circuits([1,2,3]) # optional - 4ti2
[ 0  3 -2]
[ 2 -1  0]
[ 3  0 -1]
```

directory()

Return the directory where the input files for 4ti2 are written by Sage and where 4ti2 is run.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import FourTi2
sage: f = FourTi2("/tmp/")
sage: f.directory()
'/tmp/'
```

graver(*mat=None*, *lat=None*, *project=None*)

Run the 4ti2 program `graver` on the parameters. See `http://www.4ti2.de/` for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.graver([1,2,3]) # optional - 4ti2
[ 2 -1  0]
[ 3  0 -1]
[ 1  1 -1]
[ 1 -2  1]
[ 0  3 -2]
sage: four_ti_2.graver(lat=[[1,2,3],[1,1,1]])  # optional - 4ti2
[ 1  0 -1]
[ 0  1  2]
[ 1  1  1]
[ 2  1  0]
```

groebner(*mat=None*, *lat=None*, *project=None*)

Run the 4ti2 program `groebner` on the parameters. This computes a Toric Groebner basis of a matrix. See `http://www.4ti2.de/` for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: A = [6,10,15]
sage: four_ti_2.groebner(A) # optional - 4ti2
[-5  0  2]
[-5  3  0]
sage: four_ti_2.groebner(lat=[[1,2,3],[1,1,1]]) # optional - 4ti2
[-1  0  1]
[ 2  1  0]
```

hilbert(*mat=None*, *lat=None*, *project=None*)

Run the 4ti2 program `hilbert` on the parameters. See `http://www.4ti2.de/` for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.hilbert(four_ti_2._magic3x3()) # optional - 4ti2
[2 0 1 0 1 2 1 2 0]
[1 0 2 2 1 0 0 2 1]
[0 2 1 2 1 0 1 0 2]
[1 2 0 0 1 2 2 0 1]
[1 1 1 1 1 1 1 1 1]
sage: four_ti_2.hilbert(lat=[[1,2,3],[1,1,1]])   # optional - 4ti2
[2 1 0]
[0 1 2]
[1 1 1]
```

minimize(*mat=None*, *lat=None*)

Run the 4ti2 program `minimize` on the parameters. See `http://www.4ti2.de/` for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.minimize() # optional - 4ti2
Traceback (most recent call last):
...
NotImplementedError: 4ti2 command 'minimize' not implemented in Sage.
```

ppi (*n*)

Run the 4ti2 program `ppi` on the parameters. See `http://www.4ti2.de/` for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.ppi(3) # optional - 4ti2
[-2  1  0]
[ 0 -3  2]
[-1 -1  1]
[-3  0  1]
[ 1 -2  1]
```

qsolve (*mat=None, rel=None, sign=None, project=None*)

Run the 4ti2 program `qsolve` on the parameters. See `http://www.4ti2.de/` for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: A = [[1,1,1],[1,2,3]]
sage: four_ti_2.qsolve(A) # optional - 4ti2
[[], [ 1 -2  1]]
```

rays (*mat=None, project=None*)

Run the 4ti2 program `rays` on the parameters. See `http://www.4ti2.de/` for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.rays(four_ti_2._magic3x3()) # optional - 4ti2
[0 2 1 2 1 0 1 0 2]
[1 0 2 2 1 0 0 2 1]
[1 2 0 0 1 2 2 0 1]
[2 0 1 0 1 2 1 2 0]
```

read_matrix (*filename*)

Read a matrix in 4ti2 format from the file `filename` in directory `directory()`.

INPUT:

- `filename` - The name of the file to read from.

OUTPUT: The data from the file as a matrix over **Z**.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.write_matrix([[1,2,3],[3,4,6]], "test_file")
sage: four_ti_2.read_matrix("test_file")
[1 2 3]
[3 4 6]
```

temp_project()

Return an input project file name that has not been used yet.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.temp_project()
'project_...'
```

write_array(*array*, *nrows*, *ncols*, *filename*)

Write the matrix `array` of integers (can be represented as a list of lists) to the file `filename` in directory `directory()` in 4ti2 format. The matrix must have `nrows` rows and `ncols` columns.

INPUT:

- **`array` - A matrix of integers. Can be represented as a list** of lists.
- `nrows` - The number of rows in `array`.
- `ncols` - The number of columns in `array`.
- `file` - A file name not including a path.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.write_array([[1,2,3],[3,4,5]], 2, 3, "test_file")
```

write_matrix(*mat*, *filename*)

Write the matrix `mat` to the file `filename` in 4ti2 format.

INPUT:

- **`mat` - A matrix of integers or something that can be** converted to that.
- `filename` - A file name not including a path.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.write_matrix([[1,2],[3,4]], "test_file")
```

write_single_row(*row*, *filename*)

Write the list `row` to the file `filename` in 4ti2 format as a matrix with one row.

INPUT:

- `row` - A list of integers.
- `filename` - A file name not including a path.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.write_single_row([1,2,3,4], "test_file")
```

zsolve(*mat=None*, *rel=None*, *rhs=None*, *sign=None*, *lat=None*, *project=None*)

Run the 4ti2 program `zsolve` on the parameters. See `http://www.4ti2.de/` for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: A = [[1,1,1],[1,2,3]]
sage: rel = ['<', '<']
sage: rhs = [2, 3]
sage: sign = [1,0,1]
```

```
sage: four_ti_2.zsolve(A, rel, rhs, sign) # optional - 4ti2
[
         [ 1 -1  0]
         [ 0 -1  0]
[0 0 1]  [ 0 -3  2]
[1 1 0]  [ 1 -2  1]
[0 1 0], [ 0 -2  1], []
]
sage: four_ti_2.zsolve(lat=[[1,2,3],[1,1,1]]) # optional - 4ti2
[
             [1 2 3]
[0 0 0], [], [1 1 1]
]
```

CHAPTER

FIVE

INTERFACE TO GAP

Sage provides an interface to the GAP system. This system provides extensive group theory, combinatorics, etc.

The GAP interface will only work if GAP is installed on your computer; this should be the case, since GAP is included with Sage. The interface offers three pieces of functionality:

1. `gap_console()` - A function that dumps you into an interactive command-line GAP session.
2. `gap(expr)` - Evaluation of arbitrary GAP expressions, with the result returned as a string.
3. `gap.new(expr)` - Creation of a Sage object that wraps a GAP object. This provides a Pythonic interface to GAP. For example, if `f=gap.new(10)`, then `f.Factors()` returns the prime factorization of 10 computed using GAP.

5.1 First Examples

We factor an integer using GAP:

```
sage: n = gap(20062006); n
20062006
sage: n.parent()
Gap
sage: fac = n.Factors(); fac
[ 2, 17, 59, 73, 137 ]
sage: fac.parent()
Gap
sage: fac[1]
2
```

5.2 GAP and Singular

This example illustrates conversion between Singular and GAP via Sage as an intermediate step. First we create and factor a Singular polynomial.

```
sage: singular(389)
389
sage: R1 = singular.ring(0, '(x,y)', 'dp')
sage: f = singular('9*x^16-18*x^13*y^2-9*x^12*y^3+9*x^10*y^4-18*x^11*y^2+36*x^8*y^4+18*x^7*y^5-18*x^5
sage: F = f.factorize()
sage: print F
[1]:
```

```
   _[1]=9
   _[2]=x^6-2*x^3*y^2-x^2*y^3+y^4
   _[3]=-x^5+y^2
[2]:
   1,1,2
```

Next we convert the factor $-x^5 + y^2$ to a Sage multivariate polynomial. Note that it is important to let x and y be the generators of a polynomial ring, so the eval command works.

```
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: s = F[1][3].sage_polystring(); s
'-x**5+y**2'
sage: g = eval(s); g
-x^5 + y^2
```

Next we create a polynomial ring in GAP and obtain its indeterminates:

```
sage: R = gap.PolynomialRing('Rationals', 2); R
PolynomialRing( Rationals, ["x_1", "x_2"] )
sage: I = R.IndeterminatesOfPolynomialRing(); I
[ x_1, x_2 ]
```

In order to eval g in GAP, we need to tell GAP to view the variables `x0` and `x1` as the two generators of R. This is the one tricky part. In the GAP interpreter the object `I` has its own name (which isn't `I`). We can access its name using `I.name()`.

```
sage: _ = gap.eval("x := %s[1];; y := %s[2];;"%(I.name(), I.name()))
```

Now x_0 and x_1 are defined, so we can construct the GAP polynomial f corresponding to g:

```
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: f = gap(str(g)); f
-x_1^5+x_2^2
```

We can call GAP functions on f. For example, we evaluate the GAP `Value` function, which evaluates f at the point $(1, 2)$.

```
sage: f.Value(I, [1,2])
3
sage: g(1,2)        # agrees
3
```

5.3 Saving and loading objects

Saving and loading GAP objects (using the dumps method, etc.) is *not* supported, since the output string representation of Gap objects is sometimes not valid input to GAP. Creating classes that wrap GAP objects *is* supported, via simply defining the a _gap_init_ member function that returns a string that when evaluated in GAP constructs the object. See `groups/permutation_group.py` for a nontrivial example of this.

5.4 Long Input

The GAP interface reads in even very long input (using files) in a robust manner, as long as you are creating a new object.

Note: Using `gap.eval` for long input is much less robust, and is not recommended.

```
sage: t = '"%s"'%10^10000   # ten thousand character string.
sage: a = gap(t)
```

5.5 Changing which GAP is used

Use this code to change which GAP interpreter is run. E.g.,

```
import sage.interfaces.gap
sage.interfaces.gap.gap_cmd = "/usr/local/bin/gap"
```

AUTHORS:

- David Joyner and William Stein: initial version(s)
- William Stein (2006-02-01): modified gap_console command so it uses exactly the same startup command as Gap.__init__.
- William Stein (2006-03-02): added tab completions: gap.[tab], x = gap(...), x.[tab], and docs, e.g., gap.function? and x.function?

class `sage.interfaces.gap.`**`Gap`**(*max_workspace_size=None*, *maxread=100000*, *script_subdirectory=None*, *use_workspace_cache=True*, *server=None*, *server_tmpdir=None*, *logfile=None*)

Bases: `sage.interfaces.gap.Gap_generic`

Interface to the GAP interpreter.

AUTHORS:

- William Stein and David Joyner

`console`()

Spawn a new GAP command-line session.

EXAMPLES:

```
sage: gap.console()   # not tested
*********   GAP, Version 4.5.7 of 14-Dec-2012 (free software, GPL)
*  GAP  *   http://www.gap-system.org
*********   Architecture: x86_64-unknown-linux-gnu-gcc-default64
Libs used:  gmp, readline
Loading the library and packages ...
Packages:   GAPDoc 1.5.1
Try '?help' for help. See also  '?copyright' and  '?authors'
gap>
```

`cputime`(*t=None*)

Returns the amount of CPU time that the GAP session has used. If `t` is not None, then it returns the difference between the current CPU time and `t`.

EXAMPLES:

```
sage: t = gap.cputime()
sage: t  #random
0.13600000000000001
sage: gap.Order(gap.SymmetricGroup(5))
```

```
120
sage: gap.cputime(t)   #random
0.059999999999999998
```

get(*var*, *use_file=False*)

Get the string representation of the variable var.

EXAMPLES:

```
sage: gap.set('x', '2')
sage: gap.get('x')
'2'
```

help(*s*, *pager=True*)

Print help on a given topic.

EXAMPLES:

```
sage: print gap.help('SymmetricGroup', pager=False)

50 Group Libraries

When you start GAP, it already knows several groups. Currently GAP initially
knows the following groups:
...
```

save_workspace()

Save the GAP workspace.

TESTS:

We make sure that #9938 (GAP does not start if the path to the GAP workspace file contains more than 82 characters) is fixed:

```
sage: ORIGINAL_WORKSPACE = sage.interfaces.gap.WORKSPACE
sage: sage.interfaces.gap.WORKSPACE = os.path.join(SAGE_TMP, "gap" + "0"*(80-len(SAGE_TMP)))
sage: gap = Gap()
sage: gap('3+2')   # long time (4s on sage.math, 2013)
5
sage: sage.interfaces.gap.WORKSPACE = ORIGINAL_WORKSPACE
```

set(*var*, *value*)

Set the variable var to the given value.

EXAMPLES:

```
sage: gap.set('x', '2')
sage: gap.get('x')
'2'
```

trait_names()

EXAMPLES:

```
sage: c = gap.trait_names()
sage: len(c) > 100
True
sage: 'Order' in c
True
```

class sage.interfaces.gap.**GapElement**(*parent*, *value*, *is_name=False*, *name=None*)

Bases: sage.interfaces.gap.GapElement_generic

str (*use_file=False*)
EXAMPLES:

```
sage: print gap(2)
2
```

trait_names ()
EXAMPLES:

```
sage: s5 = gap.SymmetricGroup(5)
sage: 'Centralizer' in s5.trait_names()
True
```

class sage.interfaces.gap.**GapElement_generic** (*parent*, *value*, *is_name=False*, *name=None*)
Bases: sage.interfaces.expect.ExpectElement

Generic interface to the GAP3/GAP4 interpreters.

AUTHORS:

- William Stein and David Joyner (interface for GAP4)
- Franco Saliola (Feb 2010): refactored to separate out the generic code

bool ()
EXAMPLES:

```
sage: bool(gap(2))
True
sage: gap(0).bool()
False
sage: gap('false').bool()
False
```

class sage.interfaces.gap.**GapFunction** (*parent*, *name*)
Bases: sage.interfaces.expect.ExpectFunction

class sage.interfaces.gap.**GapFunctionElement** (*obj*, *name*)
Bases: sage.interfaces.expect.FunctionElement

class sage.interfaces.gap.**Gap_generic** (*name*, *prompt*, *command=None*, *server=None*, *server_tmpdir=None*, *ulimit=None*, *maxread=100000*, *script_subdirectory=''*, *restart_on_ctrlc=False*, *verbose_start=False*, *init_code=[]*, *max_startup_time=None*, *logfile=None*, *eval_using_file_cutoff=0*, *do_cleaner=True*, *remote_cleaner=False*, *path=None*, *terminal_echo=True*)
Bases: sage.interfaces.expect.Expect

Generic interface to the GAP3/GAP4 interpreters.

AUTHORS:

- William Stein and David Joyner (interface for GAP4)
- Franco Saliola (Feb 2010): refactored to separate out the generic code

eval (*x*, *newlines=False*, *strip=True*, *split_lines=True*, ***kwds*)
Send the code in the string s to the GAP interpreter and return the output as a string.

INPUT:

- s - string containing GAP code.

- `newlines` - bool (default: True); if False, remove all backslash-newlines inserted by the GAP output formatter.
- `strip` - ignored
- `split_lines` – bool (default: True); if True then each line is evaluated separately. If False, then the whole block of code is evaluated all at once.

EXAMPLES:

```
sage: gap.eval('2+2')
'4'
sage: gap.eval('Print(4); #test\n Print(6);')
'46'
sage: gap.eval('Print("#"); Print(6);')
'#6'
sage: gap.eval('4; \n 6;')
'4\n6'
sage: gap.eval('if 3>2 then\nPrint("hi");\nfi;')
'hi'
sage: gap.eval('## this is a test\nPrint("OK")')
'OK'
sage: gap.eval('Print("This is a test. Oh no, a #");# but this is a comment\nPrint("OK")')
'This is a test. Oh no, a #OK'
sage: gap.eval('if 4>3 then')
''
sage: gap.eval('Print("Hi how are you?")')
'Hi how are you?'
sage: gap.eval('fi')
''
```

function_call (*function*, *args=None*, *kwds=None*)

Calls the GAP function with args and kwds.

EXAMPLES:

```
sage: gap.function_call('SymmetricGroup', [5])
SymmetricGroup( [ 1 .. 5 ] )
```

If the GAP function does not return a value, but prints something to the screen, then a string of the printed output is returned.

```
sage: s = gap.function_call('Display', [gap.SymmetricGroup(5).CharacterTable()])
sage: type(s)
<class 'sage.interfaces.interface.AsciiArtString'>
sage: s.startswith('CT')
True
```

get_record_element (*record*, *name*)

Return the element of a GAP record identified by `name`.

INPUT:

- `record` – a GAP record
- `name` – str

OUTPUT:

- `GapElement`

EXAMPLES:

```
sage: rec = gap('rec( a := 1, b := "2" )')
sage: gap.get_record_element(rec, 'a')
1
sage: gap.get_record_element(rec, 'b')
2
```

TESTS:

```
sage: rec = gap('rec( a := 1, b := "2" )')
sage: type(gap.get_record_element(rec, 'a'))
<class 'sage.interfaces.gap.GapElement'>
```

interrupt (*tries=None*, *timeout=1*, *quit_on_fail=True*)

Interrupt the GAP process

Gap installs a SIGINT handler, we call it directly instead of trying to sent Ctrl-C. Unlike `interrupt()`, we only try once since we are knowing what we are doing.

Sometimes GAP dies while interrupting.

EXAMPLES:

```
sage: gap._eval_line('while(1=1) do i:=1;; od;', wait_for_prompt=False);
''
sage: rc = gap.interrupt(timeout=1)
sage: [ gap(i) for i in range(10) ]   # check that it is still working
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

TESTS:

```
sage: gap('"finished computation"'); gap.interrupt(); gap('"ok"')
finished computation
True
ok
```

load_package (*pkg*, *verbose=False*)

Load the Gap package with the given name.

If loading fails, raise a RuntimeError exception.

TESTS:

```
sage: gap.load_package("chevie")
Traceback (most recent call last):
...
RuntimeError: Error loading Gap package chevie. You may want to install the gap_packages SPK
```

trait_names ()

EXAMPLES:

```
sage: c = gap.trait_names()
sage: len(c) > 100
True
sage: 'Order' in c
True
```

unbind (*var*)

Clear the variable named var.

EXAMPLES:

```
sage: gap.set('x', '2')
sage: gap.get('x')
'2'
sage: gap.unbind('x')
sage: gap.get('x')
Traceback (most recent call last):
...
RuntimeError: Gap produced error output
Error, Variable: 'x' must have a value
...
```

version()

Returns the version of GAP being used.

EXAMPLES:

```
sage: print gap.version()
4.7...
```

sage.interfaces.gap.**gap_command**(*use_workspace_cache=True*, *local=True*)

x.__init__(...) initializes x; see help(type(x)) for signature

sage.interfaces.gap.**gap_console**()

Spawn a new GAP command-line session.

Note that in gap-4.5.7 you cannot use a workspace cache that had no commandline to restore a gap session with commandline.

EXAMPLES:

```
sage: gap_console()  # not tested
*********   GAP, Version 4.5.7 of 14-Dec-2012 (free software, GPL)
*  GAP  *   http://www.gap-system.org
*********   Architecture: x86_64-unknown-linux-gnu-gcc-default64
Libs used:  gmp, readline
Loading the library and packages ...
Packages:   GAPDoc 1.5.1
Try '?help' for help. See also  '?copyright' and  '?authors'
gap>
```

TESTS:

```
sage: import subprocess
sage: from sage.interfaces.gap import gap_command
sage: cmd = 'echo "quit;" | ' + gap_command(use_workspace_cache=False)[0]
sage: gap_startup = subprocess.check_output(cmd, shell=True, stderr=subprocess.STDOUT)
sage: 'http://www.gap-system.org' in gap_startup
True
sage: 'Error' not in gap_startup
True
sage: 'sorry' not in gap_startup
True
```

sage.interfaces.gap.**gap_reset_workspace**(*max_workspace_size=None*, *verbose=False*)

Call this to completely reset the GAP workspace, which is used by default when Sage first starts GAP.

The first time you start GAP from Sage, it saves the startup state of GAP in a file `$HOME/.sage/gap/workspace-HASH`, where `HASH` is a hash of the directory where Sage is installed.

This is useful, since then subsequent startup of GAP is at least 10 times as fast. Unfortunately, if you install any new code for GAP, it won't be noticed unless you explicitly load it, e.g., with gap.load_package("my_package")

The packages sonata, guava, factint, gapdoc, grape, design, toric, and laguna are loaded in all cases before the workspace is saved, if they are available.

TESTS:

Check that `gap_reset_workspace` still works when `GAP_DIR` doesn't exist, see trac ticket #14171:

```
sage: ORIGINAL_GAP_DIR = sage.interfaces.gap.GAP_DIR
sage: ORIGINAL_WORKSPACE = sage.interfaces.gap.WORKSPACE
sage: sage.interfaces.gap.GAP_DIR = os.path.join(tmp_dir(), "test_gap_dir")
sage: sage.interfaces.gap.WORKSPACE = os.path.join(sage.interfaces.gap.GAP_DIR, "test_workspace"
sage: os.path.isfile(sage.interfaces.gap.WORKSPACE)   # long time
False
sage: gap_reset_workspace()                           # long time
sage: os.path.isfile(sage.interfaces.gap.WORKSPACE)   # long time
True
sage: sage.interfaces.gap.GAP_DIR = ORIGINAL_GAP_DIR
sage: sage.interfaces.gap.WORKSPACE = ORIGINAL_WORKSPACE
```

Check that the race condition from trac ticket #14242 has been fixed. We temporarily need to change the worksheet filename.

```
sage: ORIGINAL_WORKSPACE = sage.interfaces.gap.WORKSPACE
sage: sage.interfaces.gap.WORKSPACE = tmp_filename()
sage: from multiprocessing import Process
sage: import time
sage: gap = Gap()   # long time (reset GAP session)
sage: P = [Process(target=gap, args=("14242",)) for i in range(4)]
sage: for p in P:   # long time, indirect doctest
....:     p.start()
....:     time.sleep(0.2)
sage: for p in P:   # long time
....:     p.join()
sage: os.unlink(sage.interfaces.gap.WORKSPACE)   # long time
sage: sage.interfaces.gap.WORKSPACE = ORIGINAL_WORKSPACE
```

sage.interfaces.gap.**get_gap_memory_pool_size**()

Get the gap memory pool size for new GAP processes.

EXAMPLES:

```
sage: from sage.interfaces.gap import          ...       get_gap_memory_pool_size
sage: get_gap_memory_pool_size()    # random output
1534059315
```

sage.interfaces.gap.**gfq_gap_to_sage**(*x*, *F*)

INPUT:

- `x` - gap finite field element
- `F` - Sage finite field

OUTPUT: element of F

EXAMPLES:

```
sage: x = gap('Z(13)')
sage: F = GF(13, 'a')
sage: F(x)
2
sage: F(gap('0*Z(13)'))
0
sage: F = GF(13^2, 'a')
```

```
sage: x = gap('Z(13)')
sage: F(x)
2
sage: x = gap('Z(13^2)^3')
sage: F(x)
12*a + 11
sage: F.multiplicative_generator()^3
12*a + 11
```

AUTHOR:

- David Joyner and William Stein

`sage.interfaces.gap.intmod_gap_to_sage(x)`

INPUT:

- x – Gap integer mod ring element

EXAMPLES:

```
sage: a = gap(Mod(3, 18)); a
ZmodnZObj( 3, 18 )
sage: b = sage.interfaces.gap.intmod_gap_to_sage(a); b
3
sage: b.parent()
Ring of integers modulo 18

sage: a = gap(Mod(3, 17)); a
Z(17)
sage: b = sage.interfaces.gap.intmod_gap_to_sage(a); b
3
sage: b.parent()
Ring of integers modulo 17

sage: a = gap(Mod(0, 17)); a
0*Z(17)
sage: b = sage.interfaces.gap.intmod_gap_to_sage(a); b
0
sage: b.parent()
Ring of integers modulo 17

sage: a = gap(Mod(3, 65537)); a
ZmodpZObj( 3, 65537 )
sage: b = sage.interfaces.gap.intmod_gap_to_sage(a); b
3
sage: b.parent()
Ring of integers modulo 65537
```

`sage.interfaces.gap.is_GapElement(x)`

Returns True if x is a GapElement.

EXAMPLES:

```
sage: from sage.interfaces.gap import is_GapElement
sage: is_GapElement(gap(2))
True
sage: is_GapElement(2)
False
```

`sage.interfaces.gap.reduce_load()`

Returns an invalid GAP element. Note that this is the object returned when a GAP element is unpickled.

EXAMPLES:

```
sage: from sage.interfaces.gap import reduce_load
sage: reduce_load()
Traceback (most recent call last):
...
ValueError: The session in which this object was defined is no longer running.
sage: loads(dumps(gap(2)))
Traceback (most recent call last):
...
ValueError: The session in which this object was defined is no longer running.
```

sage.interfaces.gap.**reduce_load_GAP**()

Returns the GAP interface object defined in sage.interfaces.gap.

EXAMPLES:

```
sage: from sage.interfaces.gap import reduce_load_GAP
sage: reduce_load_GAP()
Gap
```

sage.interfaces.gap.**set_gap_memory_pool_size**(*size_in_bytes*)

Set the desired gap memory pool size.

Subsequently started GAP/libGAP instances will use this as default. Currently running instances are unchanged.

GAP will only reserve `size_in_bytes` address space. Unless you actually start a big GAP computation, the memory will not be used. However, corresponding swap space will be reserved so that GAP will always be able to use the reserved address space if needed. While nothing is actually written to disc as long as you don't run a big GAP computation, the reserved swap space will not be available for other processes.

INPUT:

- `size_in_bytes` – integer. The desired memory pool size.

EXAMPLES:

```
sage: from sage.interfaces.gap import          ...       get_gap_memory_pool_size, set_gap_memory
sage: n = get_gap_memory_pool_size()
sage: set_gap_memory_pool_size(n)
sage: n == get_gap_memory_pool_size()
True
sage: n   # random output
1534059315
```

CHAPTER

SIX

INTERFACE TO GAP3

This module implements an interface to GAP3.

AUTHORS:

- Franco Saliola (February 2010)

> **Warning:** GAP3 is not distrubuted with Sage. You need to install it separately; see the section Obtaining GAP3.

6.1 Obtaining GAP3

The GAP3 interface will only work if GAP3 is installed on your computer. Here are some ways to obtain GAP3:

- There is an optional Sage package providing GAP3 pre-packaged with several GAP3 packages:

  http://trac.sagemath.org/sage_trac/ticket/8906

- Frank Luebeck maintains a GAP3 Linux executable, optimized for i686 and statically linked for jobs of 2 GByte or more:

  http://www.math.rwth-aachen.de/~Frank.Luebeck/gap/GAP3

- Jean Michel maintains a version of GAP3 pre-packaged with CHEVIE and VKCURVE. It can be obtained here:

  http://people.math.jussieu.fr/~jmichel/chevie/chevie.html

- Finally, you can download GAP3 from the GAP website below. Since GAP3 is no longer supported, it may not be easy to install this version.

  http://www.gap-system.org/Gap3/Download3/download.html

6.2 Changing which GAP3 is used

> **Warning:** There is a bug in the pexpect module (see trac ticket #8471) that prevents the following from working correctly. For now, just make sure that `gap3` is in your `PATH`.

Sage assumes that GAP3 can be launched with the command `gap3`; that is, Sage assumes that the command `gap3` is in your `PATH`. If this is not the case, then you can start GAP3 using the following command:

```
sage: gap3 = Gap3(command='/usr/local/bin/gap3')             #not tested
```

6.3 Functionality and Examples

The interface to GAP3 offers the following functionality.

1. `gap3(expr)` - Evaluation of arbitrary GAP3 expressions, with the result returned as a Sage object wrapping the corresponding GAP3 element:

```
sage: a = gap3('3+2')                        #optional - gap3
sage: a                                      #optional - gap3
5
sage: type(a)                                #optional - gap3
<class 'sage.interfaces.gap3.GAP3Element'>

sage: S5 = gap3('SymmetricGroup(5)')         #optional - gap3
sage: S5                                     #optional - gap3
Group( (1,5), (2,5), (3,5), (4,5) )
sage: type(S5)                               #optional - gap3
<class 'sage.interfaces.gap3.GAP3Record'>
```

This provides a Pythonic interface to GAP3. If `gap_function` is the name of a GAP3 function, then the syntax `gap_element.gap_function()` returns the `gap_element` obtained by evaluating the command `gap_function(gap_element)` in GAP3:

```
sage: S5.Size()                              #optional - gap3
120
sage: S5.CharTable()                         #optional - gap3
CharTable( Group( (1,5), (2,5), (3,5), (4,5) ) )
```

Alternatively, you can instead use the syntax `gap3.gap_function(gap_element)`:

```
sage: gap3.DerivedSeries(S5)                 #optional - gap3
[ Group( (1,5), (2,5), (3,5), (4,5) ),
  Subgroup( Group( (1,5), (2,5), (3,5), (4,5) ),
            [ (1,2,5), (1,3,5), (1,4,5) ] ) ]
```

If `gap_element` corresponds to a GAP3 record, then `gap_element.recfield` provides a means to access the record element corresponding to the field `recfield`:

```
sage: S5.IsRec()                             #optional - gap3
true
sage: S5.recfields()                         #optional - gap3
['isDomain', 'isGroup', 'identity', 'generators', 'operations',
'isPermGroup', 'isFinite', '1', '2', '3', '4', 'degree']
sage: S5.identity                            #optional - gap3
()
sage: S5.degree                              #optional - gap3
5
sage: S5.1                                   #optional - gap3
(1,5)
sage: S5.2                                   #optional - gap3
(2,5)
```

2. By typing `%gap3` or `gap3.interact()` at the command-line, you can interact directly with the underlying GAP3 session.

```
sage: gap3.interact()                        #not tested

  --> Switching to Gap3 <--
```

```
gap3:
```

3. You can start a new GAP3 session as follows:

```
sage: gap3.console()                                #not tested

             ########            Lehrstuhl D fuer Mathematik
           ###    ####           RWTH Aachen
          ##         ##
         ##          #             #######            #########
        ##                        #      ##          ## #     ##
        ##           #           #       ##             #      ##
        ####        ##           ##       #             #      ##
         #####     ###           ##      ##             ##    ##
           ######### #            #########             #######
                     #                                  #
                    ##           Version 3              #
                   ###           Release 4.4            #
                  ## #           18 Apr 97              #
                 ##  #
                ##   #  Alice Niemeyer, Werner Nickel,  Martin Schoenert
               ##    #  Johannes Meier, Alex Wegner,    Thomas Bischops
              ##     #  Frank Celler,   Juergen Mnich,  Udo Polis
              ###   ##  Thomas Breuer,  Goetz Pfeiffer, Hans U. Besche
               ######   Volkmar Felsch, Heiko Theissen, Alexander Hulpke
                        Ansgar Kaup,    Akos Seress,    Erzsebet Horvath
                        Bettina Eick
                        For help enter: ?<return>
gap>
```

4. The interface also has access to the GAP3 help system:

```
sage: gap3.help('help', pager=False)                #not tested
Help _______________________________________________________...

This  section describes  together with  the following sections the   GAP
help system.  The help system lets you read the manual interactively...
```

6.4 Common Pitfalls

1. If you want to pass a string to GAP3, then you need to wrap it in single quotes as follows:

```
sage: gap3('"This is a GAP3 string"')               #optional - gap3
"This is a GAP3 string"
```

This is particularly important when a GAP3 package is loaded via the `RequirePackage` method (note that one can instead use the `load_package` method):

```
sage: gap3.RequirePackage('"chevie"')              #optional - gap3chevie
W...  to  the  CHEVIE  package, ...
```

6.5 Examples

Load a GAP3 package:

```
sage: gap3.load_package("chevie")                         #optional - gap3chevie
sage: gap3.version() # random                             #optional - gap3
'lib: v3r4p4 1997/04/18, src: v3r4p0 1994/07/10, sys: usg gcc ansi'
```

Working with GAP3 lists. Note that GAP3 lists are 1-indexed:

```
sage: L = gap3([1,2,3])                                   #optional - gap3
sage: L[1]                                                #optional - gap3
1
sage: L[2]                                                #optional - gap3
2
sage: 3 in L                                              #optional - gap3
True
sage: 4 in L                                              #optional - gap3
False
sage: m = gap3([[1,2],[3,4]])                             #optional - gap3
sage: m[2,1]                                              #optional - gap3
3
sage: [1,2] in m                                          #optional - gap3
True
sage: [3,2] in m                                          #optional - gap3
False
sage: gap3([1,2]) in m                                    #optional - gap3
True
```

Controlling variable names used by GAP3:

```
sage: gap3('2', name='x')                                 #optional - gap3
2
sage: gap3('x')                                           #optional - gap3
2
sage: gap3.unbind('x')                                    #optional - gap3
sage: gap3('x')                                           #optional - gap3
Traceback (most recent call last):
...
TypeError: Gap3 produced error output
Error, Variable: 'x' must have a value
...
```

class `sage.interfaces.gap3.`**`GAP3Element`**(*parent*, *value*, *is_name=False*, *name=None*)

Bases: `sage.interfaces.gap.GapElement_generic`

A GAP3 element

Note: If the corresponding GAP3 element is a GAP3 record, then the class is changed to a `GAP3Record`.

INPUT:

- `parent` – the GAP3 session
- `value` – the GAP3 command as a string
- `is_name` – bool (default: False); if True, then `value` is the variable name for the object
- `name` – str (default: `None`); the variable name to use for the object. If `None`, then a variable name is generated.

Note: If you pass E, X or Z for name, then an error is raised because these are sacred variable names in GAP3 that should never be redefined. Sage raises an error because GAP3 does not!

EXAMPLES:

```
sage: from sage.interfaces.gap3 import GAP3Element   #optional - gap3
sage: gap3 = Gap3()                                  #optional - gap3
sage: GAP3Element(gap3, value='3+2')                 #optional - gap3
5
sage: GAP3Element(gap3, value='sage0', is_name=True) #optional - gap3
5
```

TESTS:

```
sage: GAP3Element(gap3, value='3+2', is_name=False, name='X') #optional - gap3
Traceback (most recent call last):
...
ValueError: you are attempting to redefine X; but you should never redefine E, X or Z in gap3 (b
```

AUTHORS:

- Franco Saliola (Feb 2010)

class sage.interfaces.gap3.**GAP3Record**(*parent*, *value*, *is_name=False*, *name=None*)
Bases: sage.interfaces.gap3.GAP3Element

A GAP3 record

Note: This class should not be called directly, use GAP3Element instead. If the corresponding GAP3 element is a GAP3 record, then the class is changed to a GAP3Record.

AUTHORS:

- Franco Saliola (Feb 2010)

operations()
Return a list of the GAP3 operations for the record.

OUTPUT:

- list of strings - operations of the record

EXAMPLES:

```
sage: S5 = gap3.SymmetricGroup(5)                #optional - gap3
sage: S5.operations()                            #optional - gap3
[..., 'NormalClosure', 'NormalIntersection', 'Normalizer',
'NumberConjugacyClasses', 'PCore', 'Radical', 'SylowSubgroup',
'TrivialSubgroup', 'FusionConjugacyClasses', 'DerivedSeries', ...]
sage: S5.DerivedSeries()                         #optional - gap3
[ Group( (1,5), (2,5), (3,5), (4,5) ),
  Subgroup( Group( (1,5), (2,5), (3,5), (4,5) ),
            [ (1,2,5), (1,3,5), (1,4,5) ] ) ]
```

recfields()
Return a list of the fields for the record. (Record fields are akin to object attributes in Sage.)

OUTPUT:

- list of strings - the field records

EXAMPLES:

```
sage: S5 = gap3.SymmetricGroup(5)              #optional - gap3
sage: S5.recfields()                           #optional - gap3
['isDomain', 'isGroup', 'identity', 'generators',
 'operations', 'isPermGroup', 'isFinite', '1', '2',
 '3', '4', 'degree']
sage: S5.degree                                      #optional - gap3
5
```

trait_names()

Defines the list of methods and attributes that will appear for tab completion.

OUTPUT:

- list of strings – the available fields and operations of the record

EXAMPLES:

```
sage: S5 = gap3.SymmetricGroup(5)              #optional - gap3
sage: S5.trait_names()                         #optional - gap3
[..., 'ConjugacyClassesTry', 'ConjugateSubgroup', 'ConjugateSubgroups',
'Core', 'DegreeOperation', 'DerivedSeries', 'DerivedSubgroup',
'Difference', 'DimensionsLoewyFactors', 'DirectProduct', ...]
```

class sage.interfaces.gap3.**Gap3**(*command='gap3'*)

Bases: sage.interfaces.gap.Gap_generic

A simple Expect interface to GAP3.

EXAMPLES:

```
sage: from sage.interfaces.gap3 import Gap3
sage: gap3 = Gap3(command='gap3')
```

TESTS:

```
sage: gap3(2) == gap3(3)                       #optional - gap3
False
sage: gap3(2) == gap3(2)                       #optional - gap3
True
sage: gap3.trait_names()                       #optional - gap3
[]
```

We test the interface behaves correctly after a keyboard interrupt:

```
sage: gap3(2)                                  #optional - gap3
2
sage: try:
...     gap3._keyboard_interrupt()
... except KeyboardInterrupt:
...     pass                                   #optional - gap3
Interrupting Gap3...
sage: gap3(2)                                  #optional - gap3
2
```

We test that the interface busts out of GAP3's break loop correctly:

```
sage: f = gap3('function(L) return L[0]; end;;')   #optional - gap3
sage: f([1,2,3])                               #optional - gap3
Traceback (most recent call last):
...
RuntimeError: Gap3 produced error output
```

```
Error, List Element: <position> must be a positive integer at
return L[0] ...
```

AUTHORS:

- Franco Saliola (Feb 2010)

console()

Spawn a new GAP3 command-line session.

EXAMPLES:

```
sage: gap3.console()                               #not tested

             ########            Lehrstuhl D fuer Mathematik
           ###    ####           RWTH Aachen
          ##         ##
         ##          #             #######            #########
        ##                        #      ##          ## #     ##
        ##           #           #       ##             #      ##
        ####        ##           ##       #             #      ##
         #####     ###           ##      ##             ##    ##
           ######### #            #########             #######
                     #                                  #
                    ##           Version 3              #
                   ###           Release 4.4            #
                  ## #           18 Apr 97              #
                 ##   #
                ##    #  Alice Niemeyer, Werner Nickel,  Martin Schoenert
               ##     #  Johannes Meier, Alex Wegner,    Thomas Bischops
              ##      #  Frank Celler,   Juergen Mnich,  Udo Polis
              ###    ##  Thomas Breuer,  Goetz Pfeiffer, Hans U. Besche
               ######    Volkmar Felsch, Heiko Theissen, Alexander Hulpke
                         Ansgar Kaup,    Akos Seress,    Erzsebet Horvath
                         Bettina Eick
                         For help enter: ?<return>
gap>
```

cputime(*t=None*)

Returns the amount of CPU time that the GAP session has used in seconds. If `t` is not None, then it returns the difference between the current CPU time and `t`.

EXAMPLES:

```
sage: t = gap3.cputime()                       #optional - gap3
sage: t  #random                               #optional - gap3
0.02
sage: gap3.SymmetricGroup(5).Size()            #optional - gap3
120
sage: gap3.cputime()  #random                  #optional - gap3
0.14999999999999999
sage: gap3.cputime(t)  #random                 #optional - gap3
0.13
```

help(*topic*, *pager=True*)

Print help on the given topic.

INPUT:

- `topic` – string

EXAMPLES:

```
sage: gap3.help('help', pager=False)          #optional - gap3
Help _______________________________________________________...

This  section describes  together with  the following sectio...
help system.  The help system lets you read the manual inter...

sage: gap3.help('SymmetricGroup', pager=False) #optional - gap3
no section with this name was found
```

TESTS:

```
sage: m = gap3([[1,2,3],[4,5,6]]); m          #optional - gap3
[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]
sage: gap3.help('help', pager=False)          #optional - gap3
Help _______________________________________________________...
sage: m                                       #optional - gap3
[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]
sage: m.Print()                               #optional - gap3
[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]
sage: gap3.help('Group', pager=False)         #optional - gap3
Group ______________________________________________________...
sage: m                                       #optional - gap3
[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]
sage: m.Print()                               #optional - gap3
[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]
```

sage.interfaces.gap3.**gap3_console**()

Spawn a new GAP3 command-line session.

EXAMPLES:

```
sage: gap3.console()                               #not tested

             ########            Lehrstuhl D fuer Mathematik
           ###    ####           RWTH Aachen
          ##        ##
         ##          #             #######            #########
        ##                        #      ##          ## #     ##
        ##           #           #       ##             #      ##
        ####        ##           ##       #             #      ##
         #####     ###           ##      ##             ##    ##
           ######### #            #########             #######
                     #                                  #
                    ##           Version 3              #
                   ###           Release 4.4            #
                  ## #           18 Apr 97              #
                 ##  #
                ##   #  Alice Niemeyer, Werner Nickel,  Martin Schoenert
               ##    #  Johannes Meier, Alex Wegner,    Thomas Bischops
              ##     #  Frank Celler,   Juergen Mnich,  Udo Polis
              ###   ##  Thomas Breuer,  Goetz Pfeiffer, Hans U. Besche
               ######   Volkmar Felsch, Heiko Theissen, Alexander Hulpke
                        Ansgar Kaup,    Akos Seress,    Erzsebet Horvath
                        Bettina Eick
                        For help enter: ?<return>
gap>
```

sage.interfaces.gap3.**gap3_version**()

Return the version of GAP3 that you have in your PATH on your computer.

EXAMPLES:

```
sage: gap3_version()                                        # random, optional - gap3
'lib: v3r4p4 1997/04/18, src: v3r4p0 1994/07/10, sys: usg gcc ansi'
```

CHAPTER

SEVEN

INTERFACE TO THE GP CALCULATOR OF PARI/GP

Type `gp.[tab]` for a list of all the functions available from your Gp install. Type `gp.[tab]?` for Gp's help about a given function. Type `gp(...)` to create a new Gp object, and `gp.eval(...)` to evaluate a string using Gp (and get the result back as a string).

EXAMPLES: We illustrate objects that wrap GP objects (gp is the PARI interpreter):

```
sage: M = gp('[1,2;3,4]')
sage: M
[1, 2; 3, 4]
sage: M * M
[7, 10; 15, 22]
sage: M + M
[2, 4; 6, 8]
sage: M.matdet()
-2
```

```
sage: E = gp.ellinit([1,2,3,4,5])
sage: E.ellglobalred()
[10351, [1, -1, 0, -1], 1]
sage: E.ellan(20)
[1, 1, 0, -1, -3, 0, -1, -3, -3, -3, -1, 0, 1, -1, 0, -1, 5, -3, 4, 3]
```

```
sage: primitive_root(7)
3
sage: x = gp("znlog( Mod(2,7), Mod(3,7))")
sage: 3^x % 7
2
```

```
sage: print gp("taylor(sin(x),x)")
x - 1/6*x^3 + 1/120*x^5 - 1/5040*x^7 + 1/362880*x^9 - 1/39916800*x^11 + 1/6227020800*x^13 - 1/130767
```

GP has a powerful very efficient algorithm for numerical computation of integrals.

```
sage: gp("a = intnum(x=0,6,sin(x))")
0.03982971334963397945434770208          # 32-bit
0.039829713349633979454347702077075594548     # 64-bit
sage: gp("a")
0.03982971334963397945434770208          # 32-bit
0.039829713349633979454347702077075594548     # 64-bit
sage: gp.kill("a")
```

```
sage: gp("a")
a
```

Note that gp ASCII plots *do* work in Sage, as follows:

```
sage: print gp.eval("plot(x=0,6,sin(x))")

0.9988963 |''''''''''''_x...x_'''''''''''''''''''''''''''''''''''''''''|
          |         x"        "x                                        |
          |       _"            "_                                      |
          |       x               x                                     |
          |      "                 "                                    |
          |     "                   "                                   |
          |   _"                     "_                                 |
          |  _                         _                                |
          | _                           _                               |
          |_                             _                              |
          _                                                             |
          ````````````````````````````````"``````````````````````````````
          | " |
          | " |
          | "   "
          | "_   _"|
          | _   _ |
          | _   _ |
          | x   x |
          | "_   _" |
          | x_   _x |
-0.998955 |.."x____x".........|
          0   6
```

The GP interface reads in even very long input (using files) in a robust manner, as long as you are creating a new object.

```
sage: t = '"%s"'%10^10000   # ten thousand character string.
sage: a = gp.eval(t)
sage: a = gp(t)
```

In Sage, the PARI large Galois groups datafiles should be installed by default:

```
sage: f = gp('x^9 - x - 2')
sage: f.polgalois()
[362880, -1, 34, "S9"]
```

TESTS:

Test error recovery:

```
sage: x = gp('1/0')
Traceback (most recent call last):
...
TypeError: Error executing code in GP:
CODE:
    sage[...]=1/0;
PARI/GP ERROR:
  ***   at top-level: sage[...]=1/0
  ***                          ^--
  *** _/_: division by zero
```
```

AUTHORS:

- William Stein
- David Joyner: some examples
- William Stein (2006-03-01): added tab completion for methods: gp.[tab] and x = gp(blah); x.[tab]
- William Stein (2006-03-01): updated to work with PARI 2.2.12-beta
- William Stein (2006-05-17): updated to work with PARI 2.2.13-beta

class `sage.interfaces.gp.`**`Gp`**(*stacksize=10000000*, *maxread=100000*, *script_subdirectory=None*, *logfile=None*, *server=None*, *server_tmpdir=None*, *init_list_length=1024*)

Bases: `sage.interfaces.expect.Expect`

Interface to the PARI gp interpreter.

Type `gp.[tab]` for a list of all the functions available from your Gp install. Type `gp.[tab]?` for Gp's help about a given function. Type `gp(...)` to create a new Gp object, and `gp.eval(...)` to evaluate a string using Gp (and get the result back as a string).

INPUT:

- `stacksize` (int, default 10000000) – the initial PARI stacksize in bytes (default 10MB)
- `maxread` (int, default 100000) – ??
- `script_subdirectory` (string, default None) – name of the subdirectory of SAGE_EXTCODE/pari from which to read scripts
- `logfile` (string, default None) – log file for the pexpect interface
- `server` – name of remote server
- `server_tmpdir` – name of temporary directory on remote server
- `init_list_length` (int, default 1024) – length of initial list of local variables.

EXAMPLES:

```
sage: Gp()
PARI/GP interpreter
```

`console`()

Spawn a new GP command-line session.

EXAMPLES:

```
sage: gp.console()  # not tested
GP/PARI CALCULATOR Version 2.4.3 (development svn-12577)
amd64 running linux (x86-64/GMP-4.2.1 kernel) 64-bit version
compiled: Jul 21 2010, gcc-4.6.0 20100705 (experimental) (GCC)
(readline v6.0 enabled, extended help enabled)
```

`cputime`(*t=None*)

cputime for pari - cputime since the pari process was started.

INPUT:

- `t` - (default: None); if not None, then returns time since t

Warning: If you call gettime explicitly, e.g., gp.eval('gettime'), you will throw off this clock.

EXAMPLES:

```
sage: gp.cputime()          # random output
0.0080000000000000002
sage: gp.factor('2^157-1')
[852133201, 1; 60726444167, 1; 1654058017289, 1; 2134387368610417, 1]
sage: gp.cputime()          # random output
0.26900000000000002
```

get (*var*)

Get the value of the GP variable var.

INPUT:

- var (string) – a valid GP variable identifier

EXAMPLES:

```
sage: gp.set('x', '2')
sage: gp.get('x')
'2'
```

get_default (*var=None*)

Return the current value of a PARI gp configuration variable.

INPUT:

- var (string, default None) – the name of a PARI gp configuration variable. (See gp.default() for a list.)

OUTPUT:

(string) the value of the variable.

EXAMPLES:

```
sage: gp.get_default('log')
0
sage: gp.get_default('datadir')
'.../local/share/pari'
sage: gp.get_default('seriesprecision')
16
sage: gp.get_default('realprecision')
28              # 32-bit
38              # 64-bit
```

get_precision ()

Return the current PARI precision for real number computations.

EXAMPLES:

```
sage: gp.get_precision()
28              # 32-bit
38              # 64-bit
```

get_real_precision ()

Return the current PARI precision for real number computations.

EXAMPLES:

```
sage: gp.get_precision()
28              # 32-bit
38              # 64-bit
```

get_series_precision()

Return the current PARI power series precision.

EXAMPLES:

```
sage: gp.get_series_precision()
16
```

help(*command*)

Returns GP's help for `command`.

EXAMPLES:

```
sage: gp.help('gcd')
'gcd(x,{y}): greatest common divisor of x and y.'
```

kill(*var*)

Kill the value of the GP variable var.

INPUT:

- `var` (string) – a valid GP variable identifier

EXAMPLES:

```
sage: gp.set('xx', '22')
sage: gp.get('xx')
'22'
sage: gp.kill('xx')
sage: gp.get('xx')
'xx'
```

new_with_bits_prec(*s*, *precision=0*)

Creates a GP object from s with `precision` bits of precision. GP actually automatically increases this precision to the nearest word (i.e. the next multiple of 32 on a 32-bit machine, or the next multiple of 64 on a 64-bit machine).

EXAMPLES:

```
sage: pi_def = gp(pi); pi_def
3.141592653589793238462643383                  # 32-bit
3.1415926535897932384626433832795028842        # 64-bit
sage: pi_def.precision()
28                                             # 32-bit
38                                             # 64-bit
sage: pi_150 = gp.new_with_bits_prec(pi, 150)
sage: new_prec = pi_150.precision(); new_prec
48                                             # 32-bit
57                                             # 64-bit
sage: old_prec = gp.set_precision(new_prec); old_prec
28                                             # 32-bit
38                                             # 64-bit
sage: pi_150
3.14159265358979323846264338327950288419716939938  # 32-bit
3.14159265358979323846264338327950288419716939937510582098  # 64-bit
sage: gp.set_precision(old_prec)
48                                             # 32-bit
57                                             # 64-bit
sage: gp.get_precision()
28                                             # 32-bit
38                                             # 64-bit
```

quit (*verbose=False*, *timeout=0.25*)
Terminate the GP process.

EXAMPLES:

```
sage: a = gp('10'); a
10
sage: gp.quit()
sage: a
Traceback (most recent call last):
...
ValueError: The pari session in which this object was defined is no longer running.
sage: gp(pi)
3.1415926535897932384626433832795028842    # 64-bit
3.141592653589793238462643383              # 32-bit
```

set (*var*, *value*)
Set the GP variable var to the given value.

INPUT:

- var (string) – a valid GP variable identifier
- value – a value for the variable

EXAMPLES:

```
sage: gp.set('x', '2')
sage: gp.get('x')
'2'
```

set_default (*var=None*, *value=None*)
Set a PARI gp configuration variable, and return the old value.

INPUT:

- var (string, default None) – the name of a PARI gp configuration variable. (See gp.default() for a list.)
- value – the value to set the variable to.

EXAMPLES:

```
sage: old_prec = gp.set_default('realprecision',100); old_prec
28              # 32-bit
38              # 64-bit
sage: gp.get_default('realprecision')
100
sage: gp.set_default('realprecision',old_prec)
100
sage: gp.get_default('realprecision')
28              # 32-bit
38              # 64-bit
```

set_precision (*prec=None*)
Sets the PARI precision (in decimal digits) for real computations, and returns the old value.

EXAMPLES:

```
sage: old_prec = gp.set_precision(53); old_prec
28              # 32-bit
38              # 64-bit
sage: gp.get_precision()
53
```

```
sage: gp.set_precision(old_prec)
53
sage: gp.get_precision()
28              # 32-bit
38              # 64-bit
```

set_real_precision (*prec=None*)

Sets the PARI precision (in decimal digits) for real computations, and returns the old value.

EXAMPLES:

```
sage: old_prec = gp.set_precision(53); old_prec
28              # 32-bit
38              # 64-bit
sage: gp.get_precision()
53
sage: gp.set_precision(old_prec)
53
sage: gp.get_precision()
28              # 32-bit
38              # 64-bit
```

set_series_precision (*prec=None*)

Sets the PARI power series precision, and returns the old precision.

EXAMPLES:

```
sage: old_prec = gp.set_series_precision(50); old_prec
16
sage: gp.get_series_precision()
50
sage: gp.set_series_precision(old_prec)
50
sage: gp.get_series_precision()
16
```

trait_names ()

EXAMPLES:

```
sage: c = gp.trait_names()
sage: len(c) > 100
True
sage: 'gcd' in c
True
```

version ()

Returns the version of GP being used.

EXAMPLES:

```
sage: gp.version()  # not tested
((2, 4, 3), 'GP/PARI CALCULATOR Version 2.4.3 (development svn-12577)')
```

class sage.interfaces.gp.**GpElement** (*parent, value, is_name=False, name=None*)

Bases: sage.interfaces.expect.ExpectElement

EXAMPLES: This example illustrates dumping and loading GP elements to compressed strings.

```
sage: a = gp(39393)
sage: loads(a.dumps()) == a
True
```

Since dumping and loading uses the string representation of the object, it need not result in an identical object from the point of view of PARI:

```
sage: E = gp('ellinit([1,2,3,4,5])')
sage: loads(dumps(E)) == E
False
sage: loads(E.dumps())
[1, 2, 3, 4, 5, 9, 11, 29, 35, -183, -3429, -10351, 6128487/10351, [-1.6189099322673713423780009
[1, 2, 3, 4, 5, 9, 11, 29, 35, -183, -3429, -10351, 6128487/10351, [-1.6189099322673713423780009
sage: E
[1, 2, 3, 4, 5, 9, 11, 29, 35, -183, -3429, -10351, 6128487/10351, [-1.6189099322673713423780009
[1, 2, 3, 4, 5, 9, 11, 29, 35, -183, -3429, -10351, 6128487/10351, [-1.6189099322673713423780009
```

The two elliptic curves look the same, but internally the floating point numbers are slightly different.

bool()

EXAMPLES:

```
sage: gp(2).bool()
True
sage: bool(gp(2))
True
sage: bool(gp(0))
False
```

trait_names()

EXAMPLES:

```
sage: 'gcd' in gp(2).trait_names()
True
```

class sage.interfaces.gp.**GpFunction**(*parent*, *name*)

Bases: sage.interfaces.expect.ExpectFunction

class sage.interfaces.gp.**GpFunctionElement**(*obj*, *name*)

Bases: sage.interfaces.expect.FunctionElement

sage.interfaces.gp.**gp_console**()

Spawn a new GP command-line session.

EXAMPLES:

```
sage: gp.console()  # not tested
GP/PARI CALCULATOR Version 2.4.3 (development svn-12577)
amd64 running linux (x86-64/GMP-4.2.1 kernel) 64-bit version
compiled: Jul 21 2010, gcc-4.6.0 20100705 (experimental) (GCC)
(readline v6.0 enabled, extended help enabled)
```

sage.interfaces.gp.**gp_version**()

EXAMPLES:

```
sage: gp.version()  # not tested
((2, 4, 3), 'GP/PARI CALCULATOR Version 2.4.3 (development svn-12577)')
```

sage.interfaces.gp.**is_GpElement**(*x*)

Returns True of x is a GpElement.

EXAMPLES:

```
sage: from sage.interfaces.gp import is_GpElement
sage: is_GpElement(gp(2))
True
```

```
sage: is_GpElement(2)
False
```

sage.interfaces.gp.**reduce_load_GP**()

Returns the GP interface object defined in sage.interfaces.gp.

EXAMPLES:

```
sage: from sage.interfaces.gp import reduce_load_GP
sage: reduce_load_GP()
PARI/GP interpreter
```

CHAPTER

EIGHT

INTERFACE TO THE GNUPLOT INTERPRETER

class `sage.interfaces.gnuplot.`**`Gnuplot`**

Bases: `sage.structure.sage_object.SageObject`

Interface to the Gnuplot interpreter.

`console`()

x.__init__(...) initializes x; see help(type(x)) for signature

`gnuplot`()

x.__init__(...) initializes x; see help(type(x)) for signature

`interact`(*cmd*)

x.__init__(...) initializes x; see help(type(x)) for signature

`plot`(*cmd*, *file=None*, *verbose=True*, *reset=True*)

Draw the plot described by cmd, and possibly also save to an eps or png file.

INPUT:

- `cmd` - string
- `file` - string (default: None), if specified save plot to given file, which may be either an eps (default) or png file.
- `verbose` - print some info
- `reset` - True: reset gnuplot before making graph

OUTPUT: displays graph

Note: Note that `^` s are replaced by `**` s before being passed to gnuplot.

`plot3d`(*f*, *xmin=-1*, *xmax=1*, *ymin=-1*, *ymax=1*, *zmin=-1*, *zmax=1*, *title=None*, *samples=25*, *isosamples=20*, *xlabel='x'*, *ylabel='y'*, *interact=True*)

x.__init__(...) initializes x; see help(type(x)) for signature

`plot3d_parametric`(*f='cos(u)*(3 + v*cos(u/2)), sin(u)*(3 + v*cos(u/2)), v*sin(u/2)'*, *range1='[u=-pi:pi]'*, *range2='[v=-0.2:0.2]'*, *samples=50*, *title=None*, *interact=True*)

Draw a parametric 3d surface and rotate it interactively.

INPUT:

- `f` - (string) a function of two variables, e.g., 'cos(u)*(3 + v*cos(u/2)), sin(u)*(3 + v*cos(u/2)), v*sin(u/2)'

- range1 - (string) range of values for one variable, e.g., '[u=-pi:pi]'
- range2 - (string) range of values for another variable, e.g., '[v=-0.2:0.2]'
- samples - (int) number of sample points to use
- title - (string) title of the graph.

EXAMPLES:

```
sage: gnuplot.plot3d_parametric('v^2*sin(u), v*cos(u), v*(1-v)')   # optional - gnuplot  (nc
```

sage.interfaces.gnuplot.**gnuplot_console**()

x.__init__(...) initializes x; see help(type(x)) for signature

CHAPTER

NINE

INTERFACE TO KASH

Sage provides an interface to the KASH computer algebra system, which is a *free* (as in beer!) but *closed source* program for algebraic number theory that shares much common code with Magma. To use KASH, you must install the appropriate optional Sage package by typing something like "sage -i kash3-linux-2005.11.22" or "sage -i kash3_osx-2005.11.22". For a list of optional packages type "sage -optional". If you type one of the above commands, the (about 16MB) package will be downloaded automatically (you don't have to do that).

It is not enough to just have KASH installed on your computer. Note that the KASH Sage package is currently only available for Linux and OSX. If you need Windows, support contact me (wstein@gmail.com).

The KASH interface offers three pieces of functionality:

1. `kash_console()` - A function that dumps you into an interactive command-line KASH session. Alternatively,

   type `!kash` from the Sage prompt.

2. `kash(expr)` - Creation of a Sage object that wraps a KASH object. This provides a Pythonic interface to KASH. For example, if `f=kash.new(10)`, then `f.Factors()` returns the prime factorization of 10 computed using KASH.

3. `kash.function_name(args ...)` - Call the indicated KASH function with the given arguments are return the result as a KASH object.

4. `kash.eval(expr)` - Evaluation of arbitrary KASH expressions, with the result returned as a string.

9.1 Issues

For some reason hitting Control-C to interrupt a calculation doesn't work correctly. (TODO)

9.2 Tutorial

The examples in this tutorial require that the optional kash package be installed.

9.2.1 Basics

Basic arithmetic is straightforward. First, we obtain the result as a string.

```
sage: kash.eval('(9 - 7) * (5 + 6)')                # optional -- kash
'22'
```

Next we obtain the result as a new KASH object.

```
sage: a = kash('(9 - 7) * (5 + 6)'); a              # optional -- kash
22
sage: a.parent()                                     # optional -- kash
Kash
```

We can do arithmetic and call functions on KASH objects:

```
sage: a*a                                            # optional -- kash
484
sage: a.Factorial()                                  # optional -- kash
1124000727777607680000
```

9.2.2 Integrated Help

Use the `kash.help(name)` command to get help about a given command. This returns a list of help for each of the definitions of `name`. Use `print kash.help(name)` to nicely print out all signatures.

9.2.3 Arithmetic

Using the `kash.new` command we create Kash objects on which one can do arithmetic.

```
sage: a = kash(12345)                       # optional -- kash
sage: b = kash(25)                          # optional -- kash
sage: a/b                                   # optional -- kash
2469/5
sage: a**b                                  # optional -- kash
193765903041146393565116739165642226265776144115861523176748692334640199227714321588721871376037597656
```

9.2.4 Variable assignment

Variable assignment using `kash` is takes place in Sage.

```
sage: a = kash('32233')                     # optional -- kash
sage: a                                     # optional -- kash
32233
```

In particular, `a` is not defined as part of the KASH session itself.

```
sage: kash.eval('a')                        # optional -- kash
"Error, the variable 'a' must have a value"
```

Use `a.name()` to get the name of the KASH variable:

```
sage: a.name()                              # somewhat random; optional - kash
'sage0'
sage: kash(a.name())                        # optional -- kash
32233
```

9.2.5 Integers and Rationals

We illustrate arithmetic with integers and rationals in KASH.

```
sage: F = kash.Factorization(4352)                # optional -- kash
sage: F[1]                                        # optional -- kash
<2, 8>
sage: F[2]                                        # optional -- kash
<17, 1>
sage: F                                           # optional -- kash
[ <2, 8>, <17, 1> ], extended by:
  ext1 := 1,
  ext2 := Unassign
```

Note: For some very large numbers KASH's integer factorization seems much faster than PARI's (which is the default in Sage).

```
sage: kash.GCD(15,25)                             # optional -- kash
5
sage: kash.LCM(15,25)                             # optional -- kash
75
sage: kash.Div(25,15)                             # optional -- kash
1
sage: kash(17) % kash(5)                          # optional -- kash
2
sage: kash.IsPrime(10007)                         # optional -- kash
TRUE
sage: kash.IsPrime(2005)                          # optional -- kash
FALSE

sage: kash.NextPrime(10007)                       # optional -- kash
10009
```

9.2.6 Real and Complex Numbers

```
sage: kash.Precision()                            # optional -- kash
30
sage: kash('R')                                   # optional -- kash
Real field of precision 30
sage: kash.Precision(40)                          # optional -- kash
40
sage: kash('R')                                   # optional -- kash
Real field of precision 40
sage: z = kash('1 + 2*I')                         # optional -- kash
sage: z                                           # optional -- kash
1.000000000000000000000000000000000000000 + 2.000000000000000000000000000000000000000*I
sage: z*z                                         # optional -- kash
-3.000000000000000000000000000000000000000 + 4.000000000000000000000000000000000000000*I

sage: kash.Cos('1.24')                            # optional -- kash
0.3247962844387762365776934156973803996992
sage: kash('1.24').Cos()                          # optional -- kash
0.3247962844387762365776934156973803996992

sage: kash.Exp('1.24')                            # optional -- kash
3.455613464762675598057615494121998175400

sage: kash.Precision(30)                          # optional -- kash
```

```
30
sage: kash.Log('3+4*I')                     # optional -- kash
1.60943791243410037460075933323 + 0.927295218001612232428512462922*I
sage: kash.Log('I')                         # optional -- kash
1.57079632679489661923132169164*I

sage: kash.Sqrt(4)                          # optional -- kash
2.00000000000000000000000000000
sage: kash.Sqrt(2)                          # optional -- kash
1.41421356237309504880168872421

sage: kash.Floor('9/5')                     # optional -- kash
1
sage: kash.Floor('3/5')                     # optional -- kash
0

sage: x_c = kash('3+I')                     # optional -- kash
sage: x_c.Argument()                        # optional -- kash
0.321750554396642193401404614359
sage: x_c.Imaginary()                       # optional -- kash
1.00000000000000000000000000000
```

9.2.7 Lists

Note that list appends are completely different in KASH than in Python. Use underscore after the function name for the mutation version.

```
sage: v = kash([1,2,3]); v                 # optional -- kash
[ 1, 2, 3 ]
sage: v[1]                                 # optional -- kash
1
sage: v[3]                                 # optional -- kash
3
sage: v.Append([5])                        # optional -- kash
[ 1, 2, 3, 5 ]
sage: v                                    # optional -- kash
[ 1, 2, 3 ]
sage: v.Append_([5, 6])                    # optional -- kash
SUCCESS
sage: v                                    # optional -- kash
[ 1, 2, 3, 5, 6 ]
sage: v.Add(5)                             # optional -- kash
[ 1, 2, 3, 5, 6, 5 ]
sage: v                                    # optional -- kash
[ 1, 2, 3, 5, 6 ]
sage: v.Add_(5)                            # optional -- kash
SUCCESS
sage: v                                    # optional -- kash
[ 1, 2, 3, 5, 6, 5 ]
```

The `Apply` command applies a function to each element of a list.

:: sage: L = kash([1,2,3,4]) # optional – kash sage: L.Apply('i -> 3*i') # optional – kash [3, 6, 9, 12] sage: L # optional – kash [1, 2, 3, 4] sage: L.Apply('IsEven') # optional – kash [FALSE, TRUE, FALSE, TRUE] sage: L # optional – kash [1, 2, 3, 4]

9.2.8 Ranges

the following are examples of ranges.

```
sage: L = kash('[1..10]')                    # optional -- kash
sage: L                                      # optional -- kash
[ 1 .. 10 ]
sage: L = kash('[2,4..100]')                 # optional -- kash
sage: L                                      # optional -- kash
[ 2, 4 .. 100 ]
```

9.2.9 Sequences

9.2.10 Tuples

9.2.11 Polynomials

```
sage: f = kash('X^3 + X + 1')                # optional -- kash
sage: f + f                                  # optional -- kash
2*X^3 + 2*X + 2
sage: f * f                                  # optional -- kash
X^6 + 2*X^4 + 2*X^3 + X^2 + 2*X + 1
sage: f.Evaluate(10)                         # optional -- kash
1011
sage: Qx = kash.PolynomialAlgebra('Q')       # optional -- kash
sage: Qx.gen(1)**5 + kash('7/3')   # sage1 below somewhat random; optional -- kash
sage1.1^5 + 7/3
```

9.2.12 Number Fields

We create an equation order.

```
sage: f = kash('X^5 + 4*X^4 - 56*X^2 -16*X + 192')     # optional -- kash
sage: OK = f.EquationOrder()                           # optional -- kash
sage: OK                                               # optional -- kash
Equation Order with defining polynomial X^5 + 4*X^4 - 56*X^2 - 16*X + 192 over Z
```

```
sage: f = kash('X^5 + 4*X^4 - 56*X^2 -16*X + 192')     # optional -- kash
sage: O = f.EquationOrder()                            # optional -- kash
sage: a = O.gen(2)                                     # optional -- kash
sage: a                                                # optional -- kash
[0, 1, 0, 0, 0]
sage: O.Basis()        # output somewhat random; optional -- kash
[
_NG.1,
_NG.2,
_NG.3,
_NG.4,
_NG.5
]
sage: O.Discriminant()              # optional -- kash
1364202618880
sage: O.MaximalOrder()     # name sage2 below somewhat random; optional -- kash
Maximal Order of sage2
```

```
sage: O = kash.MaximalOrder('X^3 - 77')                 # optional -- kash
sage: I = O.Ideal(5,[2, 1, 0])                          # optional -- kash
sage: I                     # name sage14 below random; optional -- kash
Ideal of sage14
Two element generators:
[5, 0, 0]
[2, 1, 0]

sage: F = I.Factorisation()                 # optional -- kash
sage: F                     # name sage14 random; optional -- kash
[
<Prime Ideal of sage14
Two element generators:
[5, 0, 0]
[2, 1, 0], 1>
]
```

Determining whether an ideal is principal.

```
sage: I.IsPrincipal()                     # optional -- kash
FALSE, extended by:
ext1 := Unassign
```

Computation of class groups and unit groups:

```
sage: f = kash('X^5 + 4*X^4 - 56*X^2 -16*X + 192')         # optional -- kash
sage: O = kash.EquationOrder(f)                            # optional -- kash
sage: OK = O.MaximalOrder()                                # optional -- kash
sage: OK.ClassGroup()        # name sage32 below random; optional -- kash
Abelian Group isomorphic to Z/6
  Defined on 1 generator
  Relations:
  6*sage32.1 = 0, extended by:
  ext1 := Mapping from: grp^abl: sage32 to ids/ord^num: _AA

sage: U = OK.UnitGroup()                                  # optional -- kash
sage: U        # name sage34 below random; optional -- kash
Abelian Group isomorphic to Z/2 + Z + Z
  Defined on 3 generators
  Relations:
  2*sage34.1 = 0, extended by:
  ext1 := Mapping from: grp^abl: sage34 to ord^num: sage30

sage: kash.Apply('x->%s.ext1(x)'%U.name(), U.Generators().List())      # optional -- kash
[ [1, -1, 0, 0, 0], [1, 1, 0, 0, 0], [-1, 0, 0, 0, 0] ]
```

9.2.13 Function Fields

```
sage: k = kash.FiniteField(25)                                 # optional -- kash
sage: kT = k.RationalFunctionField()                           # optional -- kash
sage: kTy = kT.PolynomialAlgebra()                             # optional -- kash
sage: T = kT.gen(1)                                            # optional -- kash
sage: y = kTy.gen(1)                                           # optional -- kash
sage: f = y**3 + T**4 + 1                                      # optional -- kash
```

9.3 Long Input

The KASH interface reads in even very long input (using files) in a robust manner, as long as you are creating a new object.

Note: Using `kash.eval` for long input is much less robust, and is not recommended.

```
sage: a = kash(range(10000))                              # optional -- kash
```

Note that KASH seems to not support string or integer literals with more than 1024 digits, which is why the above example uses a list unlike for the other interfaces.

class `sage.interfaces.kash.`**`Kash`**(*max_workspace_size=None*, *maxread=100000*, *script_subdirectory=None*, *restart_on_ctrlc=True*, *logfile=None*, *server=None*, *server_tmpdir=None*)

Bases: `sage.interfaces.expect.Expect`

Interface to the Kash interpreter.

AUTHORS:

- William Stein and David Joyner

`console`()

x.__init__(...) initializes x; see help(type(x)) for signature

`eval`(*x*, *newlines=False*, *strip=True*, ***kwds*)

Send the code in the string s to the Kash interpreter and return the output as a string.

INPUT:

- `s` - string containing Kash code.
- `newlines` - bool (default: True); if False, remove all backslash-newlines inserted by the Kash output formatter.
- `strip` - ignored

`get`(*var*)

Get the value of the variable var.

`help`(*name=None*)

Return help on KASH commands.

Returns help on all commands with a given name. If name is None, return the location of the installed Kash HTML documentation.

EXAMPLES:

```
sage: X = kash.help('IntegerRing')    # optional -- kash
```

There is one entry in X for each item found in the documentation for this function: If you type `print X[0]` you will get help on about the first one, printed nicely to the screen.

AUTHORS:

- Sebastion Pauli (2006-02-04): during Sage coding sprint

`help_search`(*name*)

x.__init__(...) initializes x; see help(type(x)) for signature

set (*var*, *value*)
Set the variable var to the given value.

version ()
x.__init__(...) initializes x; see help(type(x)) for signature

class sage.interfaces.kash.**KashDocumentation**
Bases: list

x.__init__(...) initializes x; see help(type(x)) for signature

class sage.interfaces.kash.**KashElement** (*parent*, *value*, *is_name=False*, *name=None*)
Bases: sage.interfaces.expect.ExpectElement

sage.interfaces.kash.**is_KashElement** (*x*)
x.__init__(...) initializes x; see help(type(x)) for signature

sage.interfaces.kash.**kash_console** ()
x.__init__(...) initializes x; see help(type(x)) for signature

sage.interfaces.kash.**kash_version** ()
x.__init__(...) initializes x; see help(type(x)) for signature

sage.interfaces.kash.**reduce_load_Kash** ()
x.__init__(...) initializes x; see help(type(x)) for signature

CHAPTER

TEN

INTERFACE TO MAGMA

Sage provides an interface to the Magma computational algebra system. This system provides extensive functionality for number theory, group theory, combinatorics and algebra.

Note: You must have Magma installed on your computer for this interface to work. Magma is not free, so it is not included with Sage, but you can obtain it from http://magma.maths.usyd.edu.au/.

The Magma interface offers three pieces of functionality:

1. `magma_console()` - A function that dumps you into an interactive command-line Magma session.
2. `magma.new(obj)` and alternatively `magma(obj)` - Creation of a Magma object from a Sage object `obj`. This provides a Pythonic interface to Magma. For example, if `f=magma.new(10)`, then `f.Factors()` returns the prime factorization of 10 computed using Magma. If obj is a string containing an arbitrary Magma expression, then the expression is evaluated in Magma to create a Magma object. An example is `magma.new('10 div 3')`, which returns Magma integer 3.
3. `magma.eval(expr)` - Evaluation of the Magma expression `expr`, with the result returned as a string.

Type `magma.[tab]` for a list of all functions available from your Magma. Type `magma.Function?` for Magma's help about the Magma `Function`.

10.1 Parameters

Some Magma functions have optional "parameters", which are arguments that in Magma go after a colon. In Sage, you pass these using named function arguments. For example,

```
sage: E = magma('EllipticCurve([0,1,1,-1,0])')                # optional - magma
sage: E.Rank(Bound = 5)                                       # optional - magma
0
```

10.2 Multiple Return Values

Some Magma functions return more than one value. You can control how many you get using the `nvals` named parameter to a function call:

```
sage: n = magma(100)                                          # optional - magma
sage: n.IsSquare(nvals = 1)                                   # optional - magma
true
sage: n.IsSquare(nvals = 2)                                   # optional - magma
```

```
(true, 10)
sage: n = magma(-2006)                                        # optional - magma
sage: n.Factorization()                                       # optional - magma
[ <2, 1>, <17, 1>, <59, 1> ]
sage: n.Factorization(nvals=2)                                # optional - magma
([ <2, 1>, <17, 1>, <59, 1> ], -1)
```

We verify that an obviously principal ideal is principal:

```
sage: _ = magma.eval('R<x> := PolynomialRing(RationalField())')    # optional - magma
sage: O = magma.NumberField('x^2+23').MaximalOrder()               # optional - magma
sage: I = magma('ideal<%s|%s.1>'%(O.name(),O.name()))              # optional - magma
sage: I.IsPrincipal(nvals=2)                                       # optional - magma
(true, [1, 0])
```

10.3 Long Input

The Magma interface reads in even very long input (using files) in a robust manner.

```
sage: t = '"%s"'%10^10000   # ten thousand character string.       # optional - magma
sage: a = magma.eval(t)                                            # optional - magma
sage: a = magma(t)                                                 # optional - magma
```

10.4 Garbage Collection

There is a subtle point with the Magma interface, which arises from how garbage collection works. Consider the following session:

First, create a matrix m in Sage:

```
sage: m=matrix(ZZ,2,[1,2,3,4])                                     # optional - magma
```

Then I create a corresponding matrix A in Magma:

```
sage: A = magma(m)                                                 # optional - magma
```

It is called _sage_[...] in Magma:

```
sage: s = A.name(); s                                              # optional - magma
'_sage_[...]'
```

It's there:

```
sage: magma.eval(s)                                                # optional - magma
'[1 2]\n[3 4]'
```

Now I delete the reference to that matrix:

```
sage: del A                                                        # optional - magma
```

Now _sage_[...] is "zeroed out" in the Magma session:

```
sage: magma.eval(s)                                              # optional - magma
'0'
```

If Sage did not do this garbage collection, then every single time you ever create any magma object from a sage object, e.g., by doing magma(m), you would use up a lot of memory in that Magma session. This would lead to a horrible memory leak situation, which would make the Magma interface nearly useless for serious work.

10.5 Other Examples

We compute a space of modular forms with character.

```
sage: N = 20
sage: D = 20
sage: eps_top = fundamental_discriminant(D)
sage: eps = magma.KroneckerCharacter(eps_top, RationalField())       # optional - magma
sage: M2 = magma.ModularForms(eps)                                   # optional - magma
sage: print M2                                                       # optional - magma
Space of modular forms on Gamma_1(5) ...
sage: print M2.Basis()                                               # optional - magma
[
1 + 10*q^2 + 20*q^3 + 20*q^5 + 60*q^7 + ...
q + q^2 + 2*q^3 + 3*q^4 + 5*q^5 + 2*q^6 + ...
]
```

In Sage/Python (and sort of C++) coercion of an element x into a structure S is denoted by S(x). This also works for the Magma interface:

```
sage: G = magma.DirichletGroup(20)                                   # optional - magma
sage: G.AssignNames(['a', 'b'])                                      # optional - magma
sage: (G.1).Modulus()                                                # optional - magma
20
sage: e = magma.DirichletGroup(40)(G.1)                              # optional - magma
sage: print e                                                        # optional - magma
$.1
sage: print e.Modulus()                                              # optional - magma
40
```

We coerce some polynomial rings into Magma:

```
sage: R.<y> = PolynomialRing(QQ)
sage: S = magma(R)                                                   # optional - magma
sage: print S                                                        # optional - magma
Univariate Polynomial Ring in y over Rational Field
sage: S.1                                                            # optional - magma
y
```

This example illustrates that Sage doesn't magically extend how Magma implicit coercion (what there is, at least) works. The errors below are the result of Magma having a rather limited automatic coercion system compared to Sage's:

```
sage: R.<x> = ZZ[]
sage: x * 5
5*x
sage: x * 1.0
x
sage: x * (2/3)
```

```
2/3*x
sage: y = magma(x)                                                    # optional - magma
sage: y * 5                                                           # optional - magma
5*x
sage: y * 1.0                                                         # optional - magma
$.1
sage: y * (2/3)                                                       # optional - magma
Traceback (most recent call last):
...
TypeError: Error evaluating Magma code.
...
Argument types given: RngUPolElt[RngInt], FldRatElt
```

AUTHORS:

- William Stein (2005): initial version
- William Stein (2006-02-28): added extensive tab completion and interactive IPython documentation support.
- William Stein (2006-03-09): added nvals argument for magma.functions...

class `sage.interfaces.magma.`**`Magma`**(*maxread=10000*, *script_subdirectory=None*, *logfile=None*, *server=None*, *server_tmpdir=None*, *user_config=False*)

Bases: `sage.interfaces.expect.Expect`

Interface to the Magma interpreter.

Type `magma.[tab]` for a list of all the functions available from your Magma install. Type `magma.Function?` for Magma's help about a given `Function` Type `magma(...)` to create a new Magma object, and `magma.eval(...)` to run a string using Magma (and get the result back as a string).

Note: If you do not own a local copy of Magma, try using the `magma_free` command instead, which uses the free demo web interface to Magma.

EXAMPLES:

You must use nvals = 0 to call a function that doesn't return anything, otherwise you'll get an error. (nvals is the number of return values.)

```
sage: magma.SetDefaultRealFieldPrecision(200, nvals=0)  # magma >= v2.12; optional - magma
sage: magma.eval('1.1')   # optional - magma
'1.1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
sage: magma.SetDefaultRealFieldPrecision(30, nvals=0)  # optional - magma
```

`Attach`(*filename*)

Attach the given file to the running instance of Magma.

Attaching a file in Magma makes all intrinsics defined in the file available to the shell. Moreover, if the file doesn't start with the `freeze;` command, then the file is reloaded whenever it is changed. Note that functions and procedures defined in the file are *not* available. For only those, use `magma.load(filename)`.

INPUT:

- `filename` - a string

EXAMPLES: Attaching a file that exists is fine:

```
sage: SAGE_EXTCODE = SAGE_ENV['SAGE_EXTCODE']                   # optional - magma
sage: magma.attach('%s/magma/sage/basic.m'%SAGE_EXTCODE)      # optional - magma
```

Attaching a file that doesn't exist raises an exception:

```
sage: SAGE_EXTCODE = SAGE_ENV['SAGE_EXTCODE']                # optional - magma
sage: magma.attach('%s/magma/sage/basic2.m'%SAGE_EXTCODE)    # optional - magma
Traceback (most recent call last):
...
RuntimeError: Error evaluating Magma code...
```

AttachSpec (*filename*)

Attach the given spec file to the running instance of Magma.

This can attach numerous other files to the running Magma (see the Magma documentation for more details).

INPUT:

- `filename` - a string

EXAMPLES:

```
sage: SAGE_EXTCODE = SAGE_ENV['SAGE_EXTCODE']             # optional - magma
sage: magma.attach_spec('%s/magma/spec'%SAGE_EXTCODE)     # optional - magma
sage: magma.attach_spec('%s/magma/spec2'%SAGE_EXTCODE)    # optional - magma
Traceback (most recent call last):
...
RuntimeError: Can't open package spec file .../magma/spec2 for reading (No such file or dire
```

GetVerbose (*type*)

Get the verbosity level of a given algorithm class etc. in Magma.

INPUT:

- `type` - string (e.g. 'Groebner'), see Magma documentation

Note: This method is provided to be consistent with the Magma naming convention.

EXAMPLES:

```
sage: magma.SetVerbose("Groebner", 2)        # optional - magma
sage: magma.GetVerbose("Groebner")           # optional - magma
2
```

SetVerbose (*type*, *level*)

Set the verbosity level for a given algorithm class etc. in Magma.

INPUT:

- `type` - string (e.g. 'Groebner'), see Magma documentation
- `level` - integer = 0

Note: This method is provided to be consistent with the Magma naming convention.

```
sage: magma.SetVerbose("Groebner", 2)        # optional - magma
sage: magma.GetVerbose("Groebner")           # optional - magma
2
```

attach (*filename*)

Attach the given file to the running instance of Magma.

Attaching a file in Magma makes all intrinsics defined in the file available to the shell. Moreover, if the file doesn't start with the `freeze;` command, then the file is reloaded whenever it is changed. Note that functions and procedures defined in the file are *not* available. For only those, use `magma.load(filename)`.

INPUT:

- `filename` - a string

EXAMPLES: Attaching a file that exists is fine:

```
sage: SAGE_EXTCODE = SAGE_ENV['SAGE_EXTCODE']                # optional - magma
sage: magma.attach('%s/magma/sage/basic.m'%SAGE_EXTCODE)     # optional - magma
```

Attaching a file that doesn't exist raises an exception:

```
sage: SAGE_EXTCODE = SAGE_ENV['SAGE_EXTCODE']                  # optional - magma
sage: magma.attach('%s/magma/sage/basic2.m'%SAGE_EXTCODE)      # optional - magma
Traceback (most recent call last):
...
RuntimeError: Error evaluating Magma code...
```

attach_spec (*filename*)

Attach the given spec file to the running instance of Magma.

This can attach numerous other files to the running Magma (see the Magma documentation for more details).

INPUT:

- `filename` - a string

EXAMPLES:

```
sage: SAGE_EXTCODE = SAGE_ENV['SAGE_EXTCODE']             # optional - magma
sage: magma.attach_spec('%s/magma/spec'%SAGE_EXTCODE)     # optional - magma
sage: magma.attach_spec('%s/magma/spec2'%SAGE_EXTCODE)    # optional - magma
Traceback (most recent call last):
...
RuntimeError: Can't open package spec file .../magma/spec2 for reading (No such file or dire
```

bar_call (*left*, *name*, *gens*, *nvals=1*)

This is a wrapper around the Magma constructor

nameleft gens

returning nvals.

INPUT:

- `left` - something coerceable to a magma object
- `name` - name of the constructor, e.g., sub, quo, ideal, etc.
- `gens` - if a list/tuple, each item is coerced to magma; otherwise gens itself is converted to magma
- `nvals` - positive integer; number of return values

OUTPUT: a single magma object if nvals == 1; otherwise a tuple of nvals magma objects.

EXAMPLES: The bar_call function is used by the sub, quo, and ideal methods of Magma elements. Here we illustrate directly using bar_call to create quotients:

```
sage: V = magma.RModule(ZZ,3)     # optional - magma
sage: V                           # optional - magma
RModule(IntegerRing(), 3)
sage: magma.bar_call(V, 'quo', [[1,2,3]], nvals=1)   # optional - magma
```

```
RModule(IntegerRing(), 2)
sage: magma.bar_call(V, 'quo', [[1,2,3]], nvals=2)  # optional - magma
(RModule(IntegerRing(), 2),
 Mapping from: RModule(IntegerRing(), 3) to RModule(IntegerRing(), 2))
sage: magma.bar_call(V, 'quo', V, nvals=2)          # optional - magma
(RModule(IntegerRing(), 0),
 Mapping from: RModule(IntegerRing(), 3) to RModule(IntegerRing(), 0))
```

chdir(*dir*)

Change to the given directory.

INPUT:

- `dir` - string; name of a directory

EXAMPLES:

```
sage: magma.chdir('/')                 # optional - magma
sage: magma.eval('System("pwd")')      # optional - magma
'/'
```

clear(*var*)

Clear the variable named var and make it available to be used again.

INPUT:

- `var` - a string

EXAMPLES:

```
sage: magma = Magma()       # optional - magma
sage: magma.clear('foo')    # sets foo to 0 in magma; optional - magma
sage: magma.eval('foo')     # optional - magma
'0'
```

Because we cleared foo, it is set to be used as a variable name in the future:

```
sage: a = magma('10')       # optional - magma
sage: a.name()              # optional - magma
'foo'
```

The following tests that the whole variable clearing and freeing system is working correctly.

```
sage: magma = Magma()       # optional - magma
sage: a = magma('100')      # optional - magma
sage: a.name()              # optional - magma
'_sage_[1]'
sage: del a                 # optional - magma
sage: b = magma('257')      # optional - magma
sage: b.name()              # optional - magma
'_sage_[1]'
sage: del b                 # optional - magma
sage: magma('_sage_[1]')    # optional - magma
0
```

console()

Run a command line Magma session. This session is completely separate from this Magma interface.

EXAMPLES:

```
sage: magma.console()               # not tested
Magma V2.14-9     Sat Oct 11 2008 06:36:41 on one      [Seed = 1157408761]
```

```
Type ? for help.  Type <Ctrl>-D to quit.
>
Total time: 2.820 seconds, Total memory usage: 3.95MB
```

cputime (*t=None*)

Return the CPU time in seconds that has elapsed since this Magma session started. This is a floating point number, computed by Magma.

If t is given, then instead return the floating point time from when t seconds had elapsed. This is useful for computing elapsed times between two points in a running program.

INPUT:

- `t` - float (default: None); if not None, return cputime since t

OUTPUT:

- `float` - seconds

EXAMPLES:

```
sage: type(magma.cputime())          # optional - magma
<type 'float'>
sage: magma.cputime()                 # random, optional - magma
1.9399999999999999
sage: t = magma.cputime()             # optional - magma
sage: magma.cputime(t)                # random, optional - magma
0.02
```

eval (*x*, *strip=True*, ***kwds*)

Evaluate the given block x of code in Magma and return the output as a string.

INPUT:

- `x` - string of code
- `strip` - ignored

OUTPUT: string

EXAMPLES:

We evaluate a string that involves assigning to a variable and printing.

```
sage: magma.eval("a := 10;print 2+a;")       # optional - magma
'12'
```

We evaluate a large input line (note that no weird output appears and that this works quickly).

```
sage: magma.eval("a := %s;"%(10^10000))     # optional - magma
''
```

Verify that trac 9705 is fixed:

```
sage: nl=chr(10) # newline character
sage: magma.eval(  # optional - magma
... "_<x>:=PolynomialRing(Rationals());"+nl+
... "repeat"+nl+
... "  g:=3*b*x^4+18*c*x^3-6*b^2*x^2-6*b*c*x-b^3-9*c^2 where b:=Random([-10..10]) where c:=R
... "until g ne 0 and Roots(g) ne [];"+nl+
... "print "success";")
'success'
```

Verify that trac 11401 is fixed:

```
sage: nl=chr(10) # newline character
sage: magma.eval("a:=3;"+nl+"b:=5;") == nl  # optional - magma
True
sage: magma.eval("[a,b];")                  # optional - magma
'[ 3, 5 ]'
```

function_call (*function*, *args*=[ ], *params*={}, *nvals*=1)

Return result of evaluating a Magma function with given input, parameters, and asking for nvals as output.

INPUT:

- `function` - string, a Magma function name
- `args` - list of objects coercible into this magma interface
- `params` - Magma parameters, passed in after a colon
- `nvals` - number of return values from the function to ask Magma for

OUTPUT: MagmaElement or tuple of nvals MagmaElement's

EXAMPLES:

```
sage: magma.function_call('Factorization', 100)    # optional - magma
[ <2, 2>, <5, 2> ]
sage: magma.function_call('NextPrime', 100, {'Proof':False})    # optional - magma
101
sage: magma.function_call('PolynomialRing', [QQ,2])      # optional - magma
Polynomial ring of rank 2 over Rational Field
Order: Lexicographical
Variables: $.1, $.2
```

Next, we illustrate multiple return values:

```
sage: magma.function_call('IsSquare', 100)         # optional - magma
true
sage: magma.function_call('IsSquare', 100, nvals=2)     # optional - magma
(true, 10)
sage: magma.function_call('IsSquare', 100, nvals=3)     # optional - magma
Traceback (most recent call last):
...
RuntimeError: Error evaluating Magma code...
Runtime error in :=: Expected to assign 3 value(s) but only computed 2 value(s)
```

get (*var*)

Get the value of the variable var.

INPUT:

- `var` - string; name of a variable defined in the Magma session

OUTPUT:

- `string` - string representation of the value of the variable.

EXAMPLES:

```
sage: magma.set('abc', '2 + 3/5')      # optional - magma
sage: magma.get('abc')                 # optional - magma
'13/5'
```

get_verbose(*type*)

Get the verbosity level of a given algorithm class etc. in Magma.

INPUT:

- `type` - string (e.g. 'Groebner'), see Magma documentation

EXAMPLES:

```
sage: magma.set_verbose("Groebner", 2)        # optional - magma
sage: magma.get_verbose("Groebner")           # optional - magma
2
```

help(*s*)

Return Magma help on string s.

This returns what typing ?s would return in Magma.

INPUT:

- `s` - string

OUTPUT: string

EXAMPLES:

```
sage: magma.help("NextPrime")       # optional - magma
===============================================================================
PATH: /magma/ring-field-algebra/integer/prime/next-previous/NextPrime
KIND: Intrinsic
===============================================================================
NextPrime(n) : RngIntElt -> RngIntElt
NextPrime(n: parameter) : RngIntElt -> RngIntElt
...
```

ideal(*L*)

Return the Magma ideal defined by L.

INPUT:

- `L` - a list of elements of a Sage multivariate polynomial ring.

OUTPUT: The magma ideal generated by the elements of L.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: magma.ideal([x^2, y^3*x])         # optional - magma
Ideal of Polynomial ring of rank 2 over Rational Field
Order: Graded Reverse Lexicographical
Variables: x, y
Homogeneous
Basis:
[
x^2,
x*y^3
]
```

load(*filename*)

Load the file with given filename using the 'load' command in the Magma shell.

Loading a file in Magma makes all the functions and procedures in the file available. The file should not contain any intrinsics (or you'll get errors). It also runs code in the file, which can produce output.

INPUT:

•`filename` - string

OUTPUT: output printed when loading the file

EXAMPLES:

```
sage: filename = os.path.join(SAGE_TMP, 'a.m')
sage: open(filename, 'w').write('function f(n) return n^2; end function;\nprint "hi";')
sage: print magma.load(filename)      # optional - magma
Loading ".../a.m"
hi
sage: magma('f(12)')       # optional - magma
144
```

objgens (*value*, *gens*)

Create a new object with given value and gens.

INPUT:

•**value - something coercible to an element of this Magma** interface

•`gens` - string; comma separated list of variable names

OUTPUT: new Magma element that is equal to value with given gens

EXAMPLES:

```
sage: R = magma.objgens('PolynomialRing(Rationals(),2)', 'alpha,beta')     # optional - magma
sage: R.gens()          # optional - magma
[alpha, beta]
```

Because of how Magma works you can use this to change the variable names of the generators of an object:

```
sage: S = magma.objgens(R, 'X,Y')          # optional - magma
sage: R                                    # optional - magma
Polynomial ring of rank 2 over Rational Field
Order: Lexicographical
Variables: X, Y
sage: S                                    # optional - magma
Polynomial ring of rank 2 over Rational Field
Order: Lexicographical
Variables: X, Y
```

set (*var*, *value*)

Set the variable var to the given value in the Magma interpreter.

INPUT:

•`var` - string; a variable name

•`value` - string; what to set var equal to

EXAMPLES:

```
sage: magma.set('abc', '2 + 3/5')        # optional - magma
sage: magma('abc')                       # optional - magma
13/5
```

set_verbose (*type*, *level*)

Set the verbosity level for a given algorithm, class, etc. in Magma.

INPUT:

•`type` - string (e.g. 'Groebner')

- `level` - integer = 0

EXAMPLES:

```
sage: magma.set_verbose("Groebner", 2)      # optional - magma
sage: magma.get_verbose("Groebner")         # optional - magma
2
```

trait_names (*verbose=True*, *use_disk_cache=True*)

Return a list of all Magma commands.

This is used as a hook to enable custom command completion.

Magma doesn't provide any fast way to make a list of all commands, which is why caching is done by default. Note that an adverse impact of caching is that *new* commands are not picked up, e.g., user defined variables or functions.

INPUT:

- `verbose` - bool (default: True); whether to verbosely output status info the first time the command list is built
- `use_disk_cache` - bool (default: True); use cached command list, which is saved to disk.

OUTPUT: list of strings

EXAMPLES:

```
sage: len(magma.trait_names(verbose=False))    # random, optional - magma
7261
```

version ()

Return the version of Magma that you have in your PATH on your computer.

OUTPUT:

- `numbers` - 3-tuple: major, minor, etc.
- `string` - version as a string

EXAMPLES:

```
sage: magma.version()       # random, optional - magma
((2, 14, 9), 'V2.14-9')
```

class sage.interfaces.magma.**MagmaElement** (*parent*, *value*, *is_name=False*, *name=None*)

Bases: sage.interfaces.expect.ExpectElement

AssignNames (*names*)

EXAMPLES:

```
sage: G = magma.DirichletGroup(20)   # optional - magma
sage: G.AssignNames(['a','b'])       # optional - magma
sage: G.1                            # optional - magma
a

sage: G.Elements()                   # optional - magma
[
1,
a,
b,
a*b
]
```

assign_names (*names*)

EXAMPLES:

```
sage: G = magma.DirichletGroup(20)    # optional - magma
sage: G.AssignNames(['a','b'])        # optional - magma
sage: G.1                             # optional - magma
a

sage: G.Elements()                    # optional - magma
[
1,
a,
b,
a*b
]
```

eval (**args*)

Evaluate self at the inputs.

INPUT:

- `*args` - import arguments

OUTPUT: self(*args)

EXAMPLES:

```
sage: f = magma('Factorization')                # optional - magma
sage: f.evaluate(15)                            # optional - magma
[ <3, 1>, <5, 1> ]
sage: f(15)                                     # optional - magma
[ <3, 1>, <5, 1> ]
sage: f = magma('GCD')                          # optional - magma
sage: f.evaluate(15,20)                         # optional - magma
5
```

evaluate (**args*)

Evaluate self at the inputs.

INPUT:

- `*args` - import arguments

OUTPUT: self(*args)

EXAMPLES:

```
sage: f = magma('Factorization')                # optional - magma
sage: f.evaluate(15)                            # optional - magma
[ <3, 1>, <5, 1> ]
sage: f(15)                                     # optional - magma
[ <3, 1>, <5, 1> ]
sage: f = magma('GCD')                          # optional - magma
sage: f.evaluate(15,20)                         # optional - magma
5
```

gen (*n*)

Return the n-th generator of this Magma element. Note that generators are 1-based in Magma rather than 0 based!

INPUT:

- `n` - a *positive* integer

OUTPUT: MagmaElement

EXAMPLES:

```
sage: k.<a> = GF(9)
sage: magma(k).gen(1)           # optional -- magma
a
sage: R.<s,t,w> = k[]
sage: m = magma(R)              # optional -- magma
sage: m.gen(1)                  # optional -- magma
s
sage: m.gen(2)                  # optional -- magma
t
sage: m.gen(3)                  # optional -- magma
w
sage: m.gen(0)                  # optional -- magma
Traceback (most recent call last):
...
IndexError: index must be positive since Magma indexes are 1-based
sage: m.gen(4)                  # optional -- magma
Traceback (most recent call last):
...
IndexError: list index out of range
```

gen_names()

Return list of Magma variable names of the generators of self.

Note: As illustrated below, these are not the print names of the the generators of the Magma object, but special variable names in the Magma session that reference the generators.

EXAMPLES:

```
sage: R.<x,zw> = QQ[]
sage: S = magma(R)                # optional - magma
sage: S.gen_names()               # optional - magma
('_sage_[...]', '_sage_[...]')
sage: magma(S.gen_names()[1])     # optional - magma
zw
```

gens()

Return generators for self.

If self is named X is Magma, this function evaluates X.1, X.2, etc., in Magma until an error occurs. It then returns a Sage list of the resulting X.i. Note - I don't think there is a Magma command that returns the list of valid X.i. There are numerous ad hoc functions for various classes but nothing systematic. This function gets around that problem. Again, this is something that should probably be reported to the Magma group and fixed there.

AUTHORS:

- William Stein (2006-07-02)

EXAMPLES:

```
sage: magma("VectorSpace(RationalField(),3)").gens()          # optional - magma
[(1 0 0), (0 1 0), (0 0 1)]
sage: magma("AbelianGroup(EllipticCurve([1..5]))").gens()     # optional - magma
[$.1]
```

get_magma_attribute (*attrname*)

Return value of a given Magma attribute. This is like selfattrname in Magma.

OUTPUT: MagmaElement

EXAMPLES:

```
sage: V = magma("VectorSpace(RationalField(),10)")    # optional - magma
sage: V.set_magma_attribute('M','"hello"')            # optional - magma
sage: V.get_magma_attribute('M')                      # optional - magma
hello
sage: V.M                                             # optional - magma
hello
```

ideal (*gens*)

Return the ideal of self with given list of generators.

INPUT:

•gens - object or list/tuple of generators

OUTPUT:

•magma element - a Magma ideal

EXAMPLES:

```
sage: R = magma('PolynomialRing(RationalField())')         # optional - magma
sage: R.assign_names(['x'])                                # optional - magma
sage: x = R.1                                              # optional - magma
sage: R.ideal([x^2 - 1, x^3 - 1])                          # optional - magma
Ideal of Univariate Polynomial Ring in x over Rational Field generated by x - 1
```

list_attributes ()

Return the attributes of self, obtained by calling the ListAttributes function in Magma.

OUTPUT: list of strings

EXAMPLES: We observe that vector spaces in Magma have numerous funny and mysterious attributes.

```
sage: V = magma("VectorSpace(RationalField(),2)")         # optional - magma
sage: v = V.list_attributes(); v.sort(); v                # optional - magma
['Coroots', 'Involution', 'M', 'RootDatum', 'Roots', 'StrLocalData', 'T', 'decomp', 'eisen',
```

methods (*any=False*)

Return signatures of all Magma intrinsics that can take self as the first argument, as strings.

INPUT:

•any - (bool: default is False) if True, also include signatures with Any as first argument.

OUTPUT: list of strings

EXAMPLES:

```
sage: v = magma('2/3').methods()           # optional - magma
sage: v[0]                                 # optional - magma
"'*'..."
```

quo (*gens*)

Return the quotient of self by the given object or list of generators.

INPUT:

•gens - object or list/tuple of generators

OUTPUT:

- `magma element` - the quotient object
- `magma element` - mapping from self to the quotient object

EXAMPLES:

```
sage: V = magma('VectorSpace(RationalField(),3)')        # optional - magma
sage: V.quo([[1,2,3], [1,1,2]])                          # optional - magma
(Full Vector space of degree 1 over Rational Field, Mapping from: Full Vector space of degre
```

We illustrate quotienting out by an object instead of a list of generators:

```
sage: W = V.sub([ [1,2,3], [1,1,2] ])                    # optional - magma
sage: V.quo(W)                                           # optional - magma
(Full Vector space of degree 1 over Rational Field, Mapping from: Full Vector space of degre
```

We quotient a ZZ module out by a submodule.

```
sage: V = magma.RModule(ZZ,3); V # optional - magma
RModule(IntegerRing(), 3)
sage: W, phi = V.quo([[1,2,3]])  # optional - magma
sage: W                          # optional - magma
RModule(IntegerRing(), 2)
sage: phi                        # optional - magma
Mapping from: RModule(IntegerRing(), 3) to RModule(IntegerRing(), 2)
```

set_magma_attribute(*attrname*, *value*)

INPUTS: attrname - string value - something coercible to a MagmaElement

EXAMPLES:

```
sage: V = magma("VectorSpace(RationalField(),2)")    # optional - magma
sage: V.set_magma_attribute('M',10)                  # optional - magma
sage: V.get_magma_attribute('M')                     # optional - magma
10
sage: V.M                                            # optional - magma
10
```

sub(*gens*)

Return the sub-object of self with given gens.

INPUT:

- `gens` - object or list/tuple of generators

EXAMPLES:

```
sage: V = magma('VectorSpace(RationalField(),3)')        # optional - magma
sage: W = V.sub([ [1,2,3], [1,1,2] ]); W                 # optional - magma
Vector space of degree 3, dimension 2 over Rational Field
Generators:
(1 2 3)
(1 1 2)
Echelonized basis:
(1 0 1)
(0 1 1)
```

trait_names()

Return all Magma functions that have this Magma element as first input. This is used for tab completion.

Note: This function can unfortunately be slow if there are a very large number of functions, e.g., when self is an integer. (This could be fixed by the addition of an appropriate function to the Magma kernel, which is something that can only be done by the Magma developers.)

OUTPUT:

- `list` - sorted list of distinct strings

EXAMPLES:

```
sage: R.<x> = ZZ[]                          # optional - magma
sage: v = magma(R).trait_names()            # optional - magma
sage: v                                     # optional - magma
["'*'", "'+'", "'.'", "'/'", "'eq'", "'in'", "'meet'", "'subset'", ...]
```

class `sage.interfaces.magma.`**`MagmaFunction`**(*parent*, *name*)
Bases: `sage.interfaces.expect.ExpectFunction`

class `sage.interfaces.magma.`**`MagmaFunctionElement`**(*obj*, *name*)
Bases: `sage.interfaces.expect.FunctionElement`

class `sage.interfaces.magma.`**`MagmaGBLogPrettyPrinter`**(*verbosity=1*, *style='magma'*)
A device which filters Magma Groebner basis computation logs.

`flush`()
EXAMPLE:

```
sage: from sage.interfaces.magma import MagmaGBLogPrettyPrinter
sage: logs = MagmaGBLogPrettyPrinter()
sage: logs.flush()
```

`write`(*s*)
EXAMPLE:

```
sage: P.<x,y,z> = GF(32003)[]
sage: I = sage.rings.ideal.Katsura(P)
sage: _ = I.groebner_basis('magma',prot=True) # indirect doctest, optional - magma

Homogeneous weights search
...
Total Faugere F4 time: ..., real time: ...
```

`sage.interfaces.magma.`**`extcode_dir`**()
Return directory that contains all the Magma extcode. This is put in a writable directory owned by the user, since when attached, Magma has to write sig and lck files.

EXAMPLES:

```
sage: sage.interfaces.magma.extcode_dir()
'...dir_.../data/'
```

`sage.interfaces.magma.`**`is_MagmaElement`**(*x*)
Return True if x is of type MagmaElement, and False otherwise.

INPUT:

- `x` - any object

OUTPUT: bool

EXAMPLES:

```
sage: from sage.interfaces.magma import is_MagmaElement
sage: is_MagmaElement(2)
False
sage: is_MagmaElement(magma(2))                    # optional - magma
True
```

sage.interfaces.magma.**magma_console**()
Run a command line Magma session.

EXAMPLES:

```
sage: magma_console()              # not tested
Magma V2.14-9     Sat Oct 11 2008 06:36:41 on one      [Seed = 1157408761]
Type ? for help.  Type <Ctrl>-D to quit.
>
Total time: 2.820 seconds, Total memory usage: 3.95MB
```

sage.interfaces.magma.**magma_version**()
Return the version of Magma that you have in your PATH on your computer.

OUTPUT:

- `numbers` - 3-tuple: major, minor, etc.
- `string` - version as a string

EXAMPLES:

```
sage: magma_version()       # random, optional - magma
((2, 14, 9), 'V2.14-9')
```

sage.interfaces.magma.**reduce_load_Magma**()
Used in unpickling a Magma interface.

This functions just returns the global default Magma interface.

EXAMPLES:

```
sage: sage.interfaces.magma.reduce_load_Magma()
Magma
```

CHAPTER

ELEVEN

INTERFACE TO MAPLE

AUTHORS:

- William Stein (2005): maple interface
- Gregg Musiker (2006-02-02): tutorial
- William Stein (2006-03-05): added tab completion, e.g., maple.[tab], and help, e.g, maple.sin?.

You must have the optional commercial Maple interpreter installed and available as the command `maple` in your PATH in order to use this interface. You do not have to install any optional Sage packages.

Type `maple.[tab]` for a list of all the functions available from your Maple install. Type `maple.[tab]?` for Maple's help about a given function. Type `maple(...)` to create a new Maple object, and `maple.eval(...)` to run a string using Maple (and get the result back as a string).

EXAMPLES:

```
sage: maple('3 * 5')                                 # optional - maple
15
sage: maple.eval('ifactor(2005)')                    # optional - maple
'"(5)*"(401)'
sage: maple.ifactor(2005)                            # optional - maple
"(5)*"(401)
sage: maple.fsolve('x^2=cos(x)+4', 'x=0..5')         # optional - maple
1.914020619
sage: maple.factor('x^5 - y^5')                      # optional - maple
(x-y)*(x^4+x^3*y+x^2*y^2+x*y^3+y^4)
```

If the string "error" (case insensitive) occurs in the output of anything from Maple, a RuntimeError exception is raised.

11.1 Tutorial

AUTHORS:

- Gregg Musiker (2006-02-02): initial version.

This tutorial is based on the Maple Tutorial for number theory from http://www.math.mun.ca/~drideout/m3370/numtheory.html.

There are several ways to use the Maple Interface in Sage. We will discuss two of those ways in this tutorial.

1. If you have a maple expression such as

   ```
   factor( (x^5-1));
   ```

We can write that in sage as

```
sage: maple('factor(x^5-1)')                # optional - maple
(x-1)*(x^4+x^3+x^2+x+1)
```

Notice, there is no need to use a semicolon.

2. Since Sage is written in Python, we can also import maple commands and write our scripts in a Pythonic way. For example, `factor()` is a maple command, so we can also factor in Sage using

```
sage: maple('(x^5-1)').factor()             # optional - maple
(x-1)*(x^4+x^3+x^2+x+1)
```

where `expression.command()` means the same thing as `command(expression)` in Maple. We will use this second type of syntax whenever possible, resorting to the first when needed.

```
sage: maple('(x^12-1)/(x-1)').simplify()     # optional - maple
x^11+x^10+x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x+1
```

The normal command will always reduce a rational function to the lowest terms. The factor command will factor a polynomial with rational coefficients into irreducible factors over the ring of integers. So for example,

```
sage: maple('(x^12-1)').factor( )           # optional - maple
(x-1)*(x+1)*(x^2+x+1)*(x^2-x+1)*(x^2+1)*(x^4-x^2+1)
```

```
sage: maple('(x^28-1)').factor( )           # optional - maple
(x-1)*(x^6+x^5+x^4+x^3+x^2+x+1)*(x+1)*(1-x+x^2-x^3+x^4-x^5+x^6)*(x^2+1)*(x^12-x^10+x^8-x^6+x^4-x^2+1)
```

Another important feature of maple is its online help. We can access this through sage as well. After reading the description of the command, you can press q to immediately get back to your original prompt.

Incidentally you can always get into a maple console by the command

```
sage: maple.console()          # not tested
sage: !maple                   # not tested
```

Note that the above two commands are slightly different, and the first is preferred.

For example, for help on the maple command fibonacci, we type

```
sage: maple.help('fibonacci')   # not tested, since it uses a pager
```

We see there are two choices. Type

```
sage: maple.help('combinat, fibonacci')    # not tested, since it uses a pager
```

We now see how the Maple command fibonacci works under the combinatorics package. Try typing in

```
sage: maple.fibonacci(10)                # optional - maple
fibonacci(10)
```

You will get fibonacci(10) as output since Maple has not loaded the combinatorics package yet. To rectify this type

```
sage: maple('combinat[fibonacci]')(10)     # optional - maple
55
```

instead.

If you want to load the combinatorics package for future calculations, in Sage this can be done as

```
sage: maple.with_package('combinat')        # optional - maple
```

or

```
sage: maple.load('combinat')                # optional - maple
```

Now if we type `maple.fibonacci(10)`, we get the correct output:

```
sage: maple.fibonacci(10)                   # optional - maple
55
```

Some common maple packages include `combinat`, `linalg`, and `numtheory`. To produce the first 19 Fibonacci numbers, use the sequence command.

```
sage: maple('seq(fibonacci(i),i=1..19)')      # optional - maple
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181
```

Two other useful Maple commands are ifactor and isprime. For example

```
sage: maple.isprime(maple.fibonacci(27))      # optional - maple
false
sage: maple.ifactor(maple.fibonacci(27))      # optional - maple
"(2)*"(17)*"(53)*"(109)
```

Note that the isprime function that is included with Sage (which uses PARI) is better than the Maple one (it is faster and gives a provably correct answer, whereas Maple is sometimes wrong).

```
sage: alpha = maple('(1+sqrt(5))/2')          # optional - maple
sage: beta = maple('(1-sqrt(5))/2')           # optional - maple
sage: f19  = alpha^19 - beta^19/maple('sqrt(5)')       # optional - maple
sage: f19                                              # optional - maple
(1/2+1/2*5^(1/2))^19-1/5*(1/2-1/2*5^(1/2))^19*5^(1/2)
sage: f19.simplify()                # somewhat randomly ordered output; optional - maple
6765+5778/5*5^(1/2)
```

Let's say we want to write a maple program now that squares a number if it is positive and cubes it if it is negative. In maple, that would look like

```
mysqcu := proc(x)
if x > 0 then x^2;
else x^3; fi;
end;
```

In Sage, we write

```
sage: mysqcu = maple('proc(x) if x > 0 then x^2 else x^3 fi end')     # optional - maple
sage: mysqcu(5)                                                       # optional - maple
25
sage: mysqcu(-5)                                                      # optional - maple
-125
```

More complicated programs should be put in a separate file and loaded.

class `sage.interfaces.maple.`**`Maple`**(*maxread=100*, *script_subdirectory='‘*, *server=None*, *server_tmpdir=None*, *logfile=None*)

Bases: `sage.interfaces.expect.Expect`

Interface to the Maple interpreter.

Type `maple.[tab]` for a list of all the functions available from your Maple install. Type `maple.[tab]?` for Maple's help about a given function. Type `maple(...)` to create a new Maple object, and `maple.eval(...)` to run a string using Maple (and get the result back as a string).

clear(*var*)

Clear the variable named var.

Unfortunately, Maple does not have a clear command. The next best thing is to set equal to the constant 0, so that memory will be freed.

EXAMPLES:

```
sage: maple.set('xx', '2')   # optional - maple
sage: maple.get('xx')        # optional - maple
'2'
sage: maple.clear('xx')      # optional - maple
sage: maple.get('xx')        # optional - maple
'0'
```

completions(*s*)

Return all commands that complete the command starting with the string s. This is like typing s[Ctrl-T] in the maple interpreter.

EXAMPLES:

```
sage: c = maple.completions('di')  # optional - maple
sage: 'divide' in c                # optional - maple
True
```

console()

Spawn a new Maple command-line session.

EXAMPLES:

```
sage: maple.console() # not tested
    |^/|     Maple 11 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2007
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
>
```

cputime(*t=None*)

Returns the amount of CPU time that the Maple session has used. If `t` is not None, then it returns the difference between the current CPU time and `t`.

EXAMPLES:

```
sage: t = maple.cputime() # optional - maple
sage: t                   # random; optional - maple
0.02
sage: x = maple('x')      # optional - maple
sage: maple.diff(x^2, x)  # optional - maple
2*x
sage: maple.cputime(t)    # random; optional - maple
0.0
```

expect()

Returns the pexpect object for this Maple session.

EXAMPLES:

```
sage: m = Maple()
sage: m.expect() is None
True
sage: m._start()            # optional - maple
sage: m.expect()            # optional - maple
<pexpect.spawn instance at 0x...>
sage: m.quit()              # optional - maple
```

get (*var*)

Get the value of the variable var.

EXAMPLES:

```
sage: maple.set('xx', '2') # optional - maple
sage: maple.get('xx')      # optional - maple
'2'
```

help (*str*)

Display Maple help about str. This is the same as typing "?str" in the Maple console.

INPUT:

- `str` - a string to search for in the maple help system

EXAMPLES:

```
sage: maple.help('digamma') #not tested
Psi - the Digamma and Polygamma functions
...
```

load (*package*)

Make a package of Maple procedures available in the interpreter.

INPUT:

- `package` - string

EXAMPLES: Some functions are unknown to Maple until you use with to include the appropriate package.

```
sage: maple.quit()    # reset maple; optional -- maple
sage: maple('partition(10)')              # optional - maple
partition(10)
sage: maple('bell(10)')                   # optional - maple
bell(10)
sage: maple.with_package('combinat')      # optional - maple
sage: maple('partition(10)')              # optional - maple
[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 2], [1, 1, 1, 1, 1, 1, 2, 2], [1,
sage: maple('bell(10)')                   # optional - maple
115975
sage: maple('fibonacci(10)')              # optional - maple
55
```

set (*var*, *value*)

Set the variable var to the given value.

EXAMPLES:

```
sage: maple.set('xx', '2') # optional - maple
sage: maple.get('xx')      # optional - maple
'2'
```

source (*s*)
Display the Maple source (if possible) about s. This is the same as returning the output produced by the following Maple commands:

interface(verboseproc=2): print(s)

INPUT:

- s - a string representing the function whose source code you want

EXAMPLES:

```
sage: maple.source('curry')  #not tested
p -> subs('_X' = args[2 .. nargs], () -> p(_X, args))
```

trait_names (*verbose=True*, *use_disk_cache=True*)
Returns a list of all the commands defined in Maple and optionally (per default) store them to disk.

EXAMPLES:

```
sage: c = maple.trait_names(use_disk_cache=False, verbose=False) # optional - maple
sage: len(c) > 100  # optional - maple
True
sage: 'dilog' in c  # optional - maple
True
```

with_package (*package*)
Make a package of Maple procedures available in the interpreter.

INPUT:

- package - string

EXAMPLES: Some functions are unknown to Maple until you use with to include the appropriate package.

```
sage: maple.quit()   # reset maple; optional -- maple
sage: maple('partition(10)')              # optional - maple
partition(10)
sage: maple('bell(10)')                   # optional - maple
bell(10)
sage: maple.with_package('combinat')      # optional - maple
sage: maple('partition(10)')              # optional - maple
[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 2], [1, 1, 1, 1, 1, 1, 2, 2], [1,
sage: maple('bell(10)')                   # optional - maple
115975
sage: maple('fibonacci(10)')              # optional - maple
55
```

class sage.interfaces.maple.**MapleElement** (*parent*, *value*, *is_name=False*, *name=None*)
Bases: sage.interfaces.expect.ExpectElement

trait_names ()
EXAMPLES:

```
sage: a = maple(2) # optional - maple
sage: 'sin' in a.trait_names() # optional - maple
True
```

class sage.interfaces.maple.**MapleFunction** (*parent*, *name*)
Bases: sage.interfaces.expect.ExpectFunction

class sage.interfaces.maple.**MapleFunctionElement** (*obj*, *name*)
Bases: sage.interfaces.expect.FunctionElement

`sage.interfaces.maple.`**`maple_console`**`()`

Spawn a new Maple command-line session.

EXAMPLES:

```
sage: maple_console() #not tested
    |^/|     Maple 11 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2007
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
>
```

`sage.interfaces.maple.`**`reduce_load_Maple`**`()`

Returns the maple object created in sage.interfaces.maple.

EXAMPLES:

```
sage: from sage.interfaces.maple import reduce_load_Maple
sage: reduce_load_Maple()
Maple
```

CHAPTER

TWELVE

INTERFACE TO MATLAB

According to their website, MATLAB is "a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran."

The commands in this section only work if you have the "matlab" interpreter installed and available in your PATH. It's not necessary to install any special Sage packages.

EXAMPLES:

```
sage: matlab.eval('2+2')                    # optional - matlab
'\nans =\n\n     4\n'
```

```
sage: a = matlab(10)                        # optional - matlab
sage: a**10                                 # optional - matlab
   1.0000e+10
```

AUTHORS:

- William Stein (2006-10-11)

12.1 Tutorial

EXAMPLES:

```
sage: matlab('4+10')                       # optional - matlab
14
sage: matlab('date')                       # optional - matlab; random output
18-Oct-2006
sage: matlab('5*10 + 6')                   # optional - matlab
56
sage: matlab('(6+6)/3')                    # optional - matlab
4
sage: matlab('9')^2                        # optional - matlab
81
sage: a = matlab(10); b = matlab(20); c = matlab(30)    # optional - matlab
sage: avg = (a+b+c)/3 ; avg                # optional - matlab
20
sage: parent(avg)                          # optional - matlab
Matlab
```

```
sage: my_scalar = matlab('3.1415')          # optional - matlab
sage: my_scalar                             # optional - matlab
3.1415
```

```
sage: my_vector1 = matlab('[1,5,7]')     # optional - matlab
sage: my_vector1                         # optional - matlab
1     5     7
sage: my_vector2 = matlab('[1;5;7]')     # optional - matlab
sage: my_vector2                         # optional - matlab
1
5
7
sage: my_vector1 * my_vector2            # optional - matlab
75
```

```
sage: row_vector1 = matlab('[1 2 3]')             # optional - matlab
sage: row_vector2 = matlab('[3 2 1]')             # optional - matlab
sage: matrix_from_row_vec = matlab('[%s; %s]'%(row_vector1.name(), row_vector2.name()))     # optiona
sage: matrix_from_row_vec                            # optional - matlab
1     2     3
3     2     1
```

```
sage: column_vector1 = matlab('[1;3]')               # optional - matlab
sage: column_vector2 = matlab('[2;8]')               # optional - matlab
sage: matrix_from_col_vec = matlab('[%s %s]'%(column_vector1.name(), column_vector2.name()))
sage: matrix_from_col_vec                            # optional - matlab
1     2
3     8
```

```
sage: my_matrix = matlab('[8, 12, 19; 7, 3, 2; 12, 4, 23; 8, 1, 1]')    # optional - matlab
sage: my_matrix                                      # optional - matlab
     8    12    19
     7     3     2
    12     4    23
     8     1     1
```

```
sage: combined_matrix = matlab('[%s, %s]'%(my_matrix.name(), my_matrix.name()))
sage: combined_matrix                               # optional - matlab
 8    12    19     8    12    19
 7     3     2     7     3     2
12     4    23    12     4    23
 8     1     1     8     1     1
```

```
sage: tm = matlab('0.5:2:10')                       # optional - matlab
sage: tm                                            # optional - matlab
0.5000    2.5000    4.5000    6.5000    8.5000
```

```
sage: my_vector1 = matlab('[1,5,7]')                # optional - matlab
sage: my_vector1(1)                                 # optional - matlab
1
sage: my_vector1(2)                                 # optional - matlab
5
sage: my_vector1(3)                                 # optional - matlab
7
```

Matrix indexing works as follows:

```
sage: my_matrix = matlab('[8, 12, 19; 7, 3, 2; 12, 4, 23; 8, 1, 1]')     # optional - matlab
sage: my_matrix(3,2)                                # optional - matlab
4
```

Setting using parenthesis cannot work (because of how the Python language works). Use square brackets or the set function:

```
sage: my_matrix = matlab('[8, 12, 19; 7, 3, 2; 12, 4, 23; 8, 1, 1]')     # optional - matlab
sage: my_matrix.set(2,3, 1999)                                        # optional - matlab
sage: my_matrix                                                       # optional - matlab
           8          12          19
           7           3        1999
          12           4          23
           8           1           1
```

class `sage.interfaces.matlab.`**`Matlab`**(*maxread=100*, *script_subdirectory=''*, *logfile=None*, *server=None*, *server_tmpdir=None*)

Bases: `sage.interfaces.expect.Expect`

Interface to the Matlab interpreter.

EXAMPLES:

```
sage: a = matlab('[ 1, 1, 2; 3, 5, 8; 13, 21, 33 ]')     # optional - matlab
sage: b = matlab('[ 1; 3; 13]')                          # optional - matlab
sage: c = a * b                                          # optional - matlab
sage: print c                                            # optional - matlab
    30
   122
   505
```

`chdir`(*directory*)

Change MATLAB's current working directory.

EXAMPLES:

```
sage: matlab.chdir('/')            # optional - matlab
sage: matlab.pwd()                 # optional - matlab
/
```

`console`()

x.__init__(...) initializes x; see help(type(x)) for signature

`get`(*var*)

Get the value of the variable var.

EXAMPLES:

```
sage: s = matlab.eval('a = 2') # optional - matlab
sage: matlab.get('a')                # optional - matlab
'      2'
```

`sage2matlab_matrix_string`(*A*)

Return an matlab matrix from a Sage matrix.

INPUT: A Sage matrix with entries in the rationals or reals.

OUTPUT: A string that evaluates to an Matlab matrix.

EXAMPLES:

```
sage: M33 = MatrixSpace(QQ,3,3)
sage: A = M33([1,2,3,4,5,6,7,8,0])
sage: matlab.sage2matlab_matrix_string(A)    # optional - matlab
'[1, 2, 3; 4, 5, 6; 7, 8, 0]'
```

AUTHOR:

- David Joyner and William Stein

set (*var*, *value*)
Set the variable var to the given value.

strip_answer (*s*)
Returns the string s with Matlab's answer prompt removed.

EXAMPLES:

```
sage: s = '\nans =\n\n     2\n'
sage: matlab.strip_answer(s)
'     2'
```

version ()
x.__init__(...) initializes x; see help(type(x)) for signature

whos ()
x.__init__(...) initializes x; see help(type(x)) for signature

class sage.interfaces.matlab.**MatlabElement** (*parent*, *value*, *is_name=False*, *name=None*)
Bases: sage.interfaces.expect.ExpectElement

set (*i*, *j*, *x*)
x.__init__(...) initializes x; see help(type(x)) for signature

sage.interfaces.matlab.**matlab_console** ()
This requires that the optional matlab program be installed and in your PATH, but no optional Sage packages need be installed.

EXAMPLES:

```
sage: matlab_console()                   # optional - matlab; not tested
                                      < M A T L A B >
                      Copyright 1984-2006 The MathWorks, Inc.
...
>> 2+3
```

ans =

5

quit

Typing quit exits the matlab console and returns you to Sage. matlab, like Sage, remembers its history from one session to another.

sage.interfaces.matlab.**matlab_version** ()
Return the version of Matlab installed.

EXAMPLES:

```
sage: matlab_version()    # random; optional - matlab
'7.2.0.283 (R2006a)'
```

sage.interfaces.matlab.**reduce_load_Matlab** ()
x.__init__(...) initializes x; see help(type(x)) for signature

CHAPTER

THIRTEEN

PEXPECT INTERFACE TO MAXIMA

Maxima is a free GPL'd general purpose computer algebra system whose development started in 1968 at MIT. It contains symbolic manipulation algorithms, as well as implementations of special functions, including elliptic functions and generalized hypergeometric functions. Moreover, Maxima has implementations of many functions relating to the invariant theory of the symmetric group S_n. (However, the commands for group invariants, and the corresponding Maxima documentation, are in French.) For many links to Maxima documentation see http://maxima.sourceforge.net/documentation.html.

AUTHORS:

- William Stein (2005-12): Initial version
- David Joyner: Improved documentation
- William Stein (2006-01-08): Fixed bug in parsing
- William Stein (2006-02-22): comparisons (following suggestion of David Joyner)
- William Stein (2006-02-24): *greatly* improved robustness by adding sequence numbers to IO bracketing in _eval_line
- Robert Bradshaw, Nils Bruin, Jean-Pierre Flori (2010,2011): Binary library interface

This is the interface used by the maxima object:

```
sage: type(maxima)
<class 'sage.interfaces.maxima.Maxima'>
```

If the string "error" (case insensitive) occurs in the output of anything from Maxima, a RuntimeError exception is raised.

EXAMPLES: We evaluate a very simple expression in Maxima.

```
sage: maxima('3 * 5')
15
```

We factor $x^5 - y^5$ in Maxima in several different ways. The first way yields a Maxima object.

```
sage: F = maxima.factor('x^5 - y^5')
sage: F
-(y-x)*(y^4+x*y^3+x^2*y^2+x^3*y+x^4)
sage: type(F)
<class 'sage.interfaces.maxima.MaximaElement'>
```

Note that Maxima objects can also be displayed using "ASCII art"; to see a normal linear representation of any Maxima object x. Just use the print command: use `str(x)`.

```
sage: print F
                           4       3     2  2     3       4
               - (y - x) (y  + x y  + x  y  + x  y + x )
```

You can always use `repr(x)` to obtain the linear representation of an object. This can be useful for moving maxima data to other systems.

```
sage: repr(F)
'-(y-x)*(y^4+x*y^3+x^2*y^2+x^3*y+x^4)'
sage: F.str()
'-(y-x)*(y^4+x*y^3+x^2*y^2+x^3*y+x^4)'
```

The `maxima.eval` command evaluates an expression in maxima and returns the result as a *string* not a maxima object.

```
sage: print maxima.eval('factor(x^5 - y^5)')
-(y-x)*(y^4+x*y^3+x^2*y^2+x^3*y+x^4)
```

We can create the polynomial f as a Maxima polynomial, then call the factor method on it. Notice that the notation `f.factor()` is consistent with how the rest of Sage works.

```
sage: f = maxima('x^5 - y^5')
sage: f^2
(x^5-y^5)^2
sage: f.factor()
-(y-x)*(y^4+x*y^3+x^2*y^2+x^3*y+x^4)
```

Control-C interruption works well with the maxima interface, because of the excellent implementation of maxima. For example, try the following sum but with a much bigger range, and hit control-C.

```
sage: maxima('sum(1/x^2, x, 1, 10)')
1968329/1270080
```

13.1 Tutorial

We follow the tutorial at http://maxima.sourceforge.net/docs/intromax/intromax.html.

```
sage: maxima('1/100 + 1/101')
201/10100
```

```
sage: a = maxima('(1 + sqrt(2))^5'); a
(sqrt(2)+1)^5
sage: a.expand()
29*sqrt(2)+41
```

```
sage: a = maxima('(1 + sqrt(2))^5')
sage: float(a)
82.01219330881975
sage: a.numer()
82.01219330881975
```

```
sage: maxima.eval('fpprec : 100')
'100'
```

```
sage: a.bfloat()
8.20121933088197564152489730020812442785204843859314941221237124017312418754011041266612384955016056b
```

```
sage: maxima('100!')
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286
```

```
sage: f = maxima('(x + 3*y + x^2*y)^3')
sage: f.expand()
x^6*y^3+9*x^4*y^3+27*x^2*y^3+27*y^3+3*x^5*y^2+18*x^3*y^2+27*x*y^2+3*x^4*y+9*x^2*y+x^3
sage: f.subst('x=5/z')
(5/z+25*y/z^2+3*y)^3
sage: g = f.subst('x=5/z')
sage: h = g.ratsimp(); h
(27*y^3*z^6+135*y^2*z^5+(675*y^3+225*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^2+9375*y^2*z+15625
sage: h.factor()
(3*y*z^2+5*z+25*y)^3/z^6
```

```
sage: eqn = maxima(['a+b*c=1', 'b-a*c=0', 'a+b=5'])
sage: s = eqn.solve('[a,b,c]'); s
[[a=(25*sqrt(79)*%i+25)/(6*sqrt(79)*%i-34),b=(5*sqrt(79)*%i+5)/(sqrt(79)*%i+11),c=(sqrt(79)*%i+1)/10]
```

Here is an example of solving an algebraic equation:

```
sage: maxima('x^2+y^2=1').solve('y')
[y=-sqrt(1-x^2),y=sqrt(1-x^2)]
sage: maxima('x^2 + y^2 = (x^2 - y^2)/sqrt(x^2 + y^2)').solve('y')
[y=-sqrt((-y^2-x^2)*sqrt(y^2+x^2)+x^2),y=sqrt((-y^2-x^2)*sqrt(y^2+x^2)+x^2)]
```

You can even nicely typeset the solution in latex:

```
sage: latex(s)
\left[ \left[ a={{25\,\sqrt{79}\,i+25}\over{6\,\sqrt{79}\,i-34}} ,   b={{5\,\sqrt{79}\,i+5}\over{\sqr
```

To have the above appear onscreen via `xdvi`, type `view(s)`. (TODO: For OS X should create pdf output and use preview instead?)

```
sage: e = maxima('sin(u + v) * cos(u)^3'); e
cos(u)^3*sin(v+u)
sage: f = e.trigexpand(); f
cos(u)^3*(cos(u)*sin(v)+sin(u)*cos(v))
sage: f.trigreduce()
(sin(v+4*u)+sin(v-2*u))/8+(3*sin(v+2*u)+3*sin(v))/8
sage: w = maxima('3 + k*%i')
sage: f = w^2 + maxima('%e')^w
sage: f.realpart()
%e^3*cos(k)-k^2+9
```

```
sage: f = maxima('x^3 * %e^(k*x) * sin(w*x)'); f
x^3*%e^(k*x)*sin(w*x)
sage: f.diff('x')
k*x^3*%e^(k*x)*sin(w*x)+3*x^2*%e^(k*x)*sin(w*x)+w*x^3*%e^(k*x)*cos(w*x)
sage: f.integrate('x')
(((k*w^6+3*k^3*w^4+3*k^5*w^2+k^7)*x^3+(3*w^6+3*k^2*w^4-3*k^4*w^2-3*k^6)*x^2+(-18*k*w^4-12*k^3*w^2+6*k
```

```
sage: f = maxima('1/x^2')
sage: f.integrate('x', 1, 'inf')
```

```
1
sage: g = maxima('f/sinh(k*x)^4')
sage: g.taylor('x', 0, 3)
f/(k^4*x^4)-2*f/(3*k^2*x^2)+11*f/45-62*k^2*f*x^2/945

sage: maxima.taylor('asin(x)','x',0, 10)
x+x^3/6+3*x^5/40+5*x^7/112+35*x^9/1152
```

13.2 Examples involving matrices

We illustrate computing with the matrix whose i, j entry is i/j, for $i, j = 1, \ldots, 4$.

```
sage: f = maxima.eval('f[i,j] := i/j')
sage: A = maxima('genmatrix(f,4,4)'); A
matrix([1,1/2,1/3,1/4],[2,1,2/3,1/2],[3,3/2,1,3/4],[4,2,4/3,1])
sage: A.determinant()
0
sage: A.echelon()
matrix([1,1/2,1/3,1/4],[0,0,0,0],[0,0,0,0],[0,0,0,0])
sage: A.eigenvalues()
[[0,4],[3,1]]
sage: A.eigenvectors()
[[[0,4],[3,1]],[[[1,0,0,-4],[0,1,0,-2],[0,0,1,-4/3]],[[1,2,3,4]]]]
```

We can also compute the echelon form in Sage:

```
sage: B = matrix(QQ, A)
sage: B.echelon_form()
[  1 1/2 1/3 1/4]
[  0   0   0   0]
[  0   0   0   0]
[  0   0   0   0]
sage: B.charpoly('x').factor()
(x - 4) * x^3
```

13.3 Laplace Transforms

We illustrate Laplace transforms:

```
sage: _ = maxima.eval("f(t) := t*sin(t)")
sage: maxima("laplace(f(t),t,s)")
2*s/(s^2+1)^2

sage: maxima("laplace(delta(t-3),t,s)") #Dirac delta function
%e^-(3*s)

sage: _ = maxima.eval("f(t) := exp(t)*sin(t)")
sage: maxima("laplace(f(t),t,s)")
1/(s^2-2*s+2)
```

```
sage: _ = maxima.eval("f(t) := t^5*exp(t)*sin(t)")
sage: maxima("laplace(f(t),t,s)")
360*(2*s-2)/(s^2-2*s+2)^4-480*(2*s-2)^3/(s^2-2*s+2)^5+120*(2*s-2)^5/(s^2-2*s+2)^6
sage: print maxima("laplace(f(t),t,s)")
                                         3                 5
           360 (2 s - 2)    480 (2 s - 2)     120 (2 s - 2)
          --------------- - --------------- + ---------------
            2           4     2           5     2           6
          (s  - 2 s + 2)    (s  - 2 s + 2)    (s  - 2 s + 2)
```

```
sage: maxima("laplace(diff(x(t),t),t,s)")
s*'laplace(x(t),t,s)-x(0)
```

```
sage: maxima("laplace(diff(x(t),t,2),t,s)")
-?%at('diff(x(t),t,1),t=0)+s^2*'laplace(x(t),t,s)-x(0)*s
```

It is difficult to read some of these without the 2d representation:

```
sage: print maxima("laplace(diff(x(t),t,2),t,s)")
                     !
            d        !         2
          - -- (x(t))!      + s  laplace(x(t), t, s) - x(0) s
            dt       !
                     !t = 0
```

Even better, use `view(maxima("laplace(diff(x(t),t,2),t,s"))` to see a typeset version.

13.4 Continued Fractions

A continued fraction $a + 1/(b + 1/(c + \cdots))$ is represented in maxima by the list $[a, b, c, \ldots]$.

```
sage: maxima("cf((1 + sqrt(5))/2)")
[1,1,1,1,2]
sage: maxima("cf ((1 + sqrt(341))/2)")
[9,1,2,1,2,1,17,1,2,1,2,1,17,1,2,1,2,1,17,2]
```

13.5 Special examples

In this section we illustrate calculations that would be awkward to do (as far as I know) in non-symbolic computer algebra systems like MAGMA or GAP.

We compute the gcd of $2x^{n+4} - x^{n+2}$ and $4x^{n+1} + 3x^n$ for arbitrary n.

```
sage: f = maxima('2*x^(n+4) - x^(n+2)')
sage: g = maxima('4*x^(n+1) + 3*x^n')
sage: f.gcd(g)
x^n
```

You can plot 3d graphs (via gnuplot):

```
sage: maxima('plot3d(x^2-y^2, [x,-2,2], [y,-2,2], [grid,12,12])')  # not tested
[displays a 3 dimensional graph]
```

You can formally evaluate sums (note the `nusum` command):

```
sage: S = maxima('nusum(exp(1+2*i/n),i,1,n)')
sage: print S
                        2/n + 3                   2/n + 1
                      %e                        %e
              ----------------------- - -----------------------
                 1/n         1/n           1/n         1/n
              (%e    - 1) (%e    + 1)   (%e    - 1) (%e    + 1)
```

We formally compute the limit as $n \to \infty$ of $2S/n$ as follows:

```
sage: T = S*maxima('2/n')
sage: T.tlimit('n','inf')
%e^3-%e
```

13.6 Miscellaneous

Obtaining digits of π:

```
sage: maxima.eval('fpprec : 100')
'100'
sage: maxima(pi).bfloat()
3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117068
```

Defining functions in maxima:

```
sage: maxima.eval('fun[a] := a^2')
'fun[a]:=a^2'
sage: maxima('fun[10]')
100
```

13.7 Interactivity

Unfortunately maxima doesn't seem to have a non-interactive mode, which is needed for the Sage interface. If any Sage call leads to maxima interactively answering questions, then the questions can't be answered and the maxima session may hang. See the discussion at http://www.ma.utexas.edu/pipermail/maxima/2005/011061.html for some ideas about how to fix this problem. An example that illustrates this problem is `maxima.eval('integrate (exp(a*x), x, 0, inf)')`.

13.8 Latex Output

To TeX a maxima object do this:

```
sage: latex(maxima('sin(u) + sinh(v^2)'))
\sinh v^2+\sin u
```

Here's another example:

```
sage: g = maxima('exp(3*%i*x)/(6*%i) + exp(%i*x)/(2*%i) + c')
sage: latex(g)
-{{i\,e^{3\,i\,x}}\over{6}}-{{i\,e^{i\,x}}\over{2}}+c
```

13.9 Long Input

The MAXIMA interface reads in even very long input (using files) in a robust manner, as long as you are creating a new object.

Note: Using `maxima.eval` for long input is much less robust, and is not recommended.

```
sage: t = '"%s"'%10^10000   # ten thousand character string.
sage: a = maxima(t)
```

TESTS: This working tests that a subtle bug has been fixed:

```
sage: f = maxima.function('x','gamma(x)')
sage: g = f(1/7)
sage: g
gamma(1/7)
sage: del f
sage: maxima(sin(x))
sin(_SAGE_VAR_x)
```

This tests to make sure we handle the case where Maxima asks if an expression is positive or zero.

```
sage: var('Ax,Bx,By')
(Ax, Bx, By)
sage: t = -Ax*sin(sqrt(Ax^2)/2)/(sqrt(Ax^2)*sqrt(By^2 + Bx^2))
sage: t.limit(Ax=0, dir='+')
0
```

A long complicated input expression:

```
sage: maxima._eval_line('((((((((((0) + ((1) / ((n0) ^ (0)))) + ((1) / ((n1) ^ (1)))) + ((1) / ((n2)
'1/n9^9+1/n8^8+1/n7^7+1/n6^6+1/n5^5+1/n4^4+1/n3^3+1/n2^2+1/n1+1'
```

class `sage.interfaces.maxima.`**`Maxima`**(*script_subdirectory=None*, *logfile=None*, *server=None*, *init_code=None*)

Bases: `sage.interfaces.maxima_abstract.MaximaAbstract`, `sage.interfaces.expect.Expect`

Interface to the Maxima interpreter.

EXAMPLES:

```
sage: m = Maxima()
sage: m == maxima
False
```

`clear`(*var*)

Clear the variable named var.

EXAMPLES:

```
sage: maxima.set('xxxxx', '2')
sage: maxima.get('xxxxx')
'2'
sage: maxima.clear('xxxxx')
sage: maxima.get('xxxxx')
'xxxxx'
```

get (*var*)

Get the string value of the variable var.

EXAMPLES:

```
sage: maxima.set('xxxxx', '2')
sage: maxima.get('xxxxx')
'2'
```

lisp (*cmd*)

Send a lisp command to Maxima.

Note: The output of this command is very raw - not pretty.

EXAMPLES:

```
sage: maxima.lisp("(+ 2 17)")   # random formatted output
 :lisp (+ 2 17)
19
(
```

set (*var*, *value*)

Set the variable var to the given value.

INPUT:

- `var` - string
- `value` - string

EXAMPLES:

```
sage: maxima.set('xxxxx', '2')
sage: maxima.get('xxxxx')
'2'
```

class `sage.interfaces.maxima.`**MaximaElement** (*parent*, *value*, *is_name=False*, *name=None*)

Bases: `sage.interfaces.maxima_abstract.MaximaAbstractElement`, `sage.interfaces.expect.ExpectElement`

Element of Maxima through Pexpect interface.

EXAMPLES:

Elements of this class should not be created directly. The targeted parent should be used instead:

```
sage: maxima(3)
3
sage: maxima(cos(x)+e^234)
cos(_SAGE_VAR_x)+%e^234
```

display2d (*onscreen=True*)

Return the 2d string representation of this Maxima object.

EXAMPLES:

```
sage: F = maxima('x^5 - y^5').factor()
sage: F.display2d()
                           4        3      2  2      3        4
               - (y - x) (y   + x y   + x  y   + x  y  + x )
```

class `sage.interfaces.maxima.`**`MaximaElementFunction`**(*parent*, *name*, *defn*, *args*, *latex*)

Bases: `sage.interfaces.maxima.MaximaElement`, `sage.interfaces.maxima_abstract.MaximaAbstrac`

Maxima user-defined functions.

EXAMPLES:

Elements of this class should not be created directly. The method `function` of the targeted parent should be used instead:

```
sage: maxima.function('x,y','h(x)*y')
h(x)*y
```

class `sage.interfaces.maxima.`**`MaximaFunction`**(*parent*, *name*)

Bases: `sage.interfaces.maxima_abstract.MaximaAbstractFunction`, `sage.interfaces.expect.ExpectFunction`

class `sage.interfaces.maxima.`**`MaximaFunctionElement`**(*obj*, *name*)

Bases: `sage.interfaces.maxima_abstract.MaximaAbstractFunctionElement`, `sage.interfaces.expect.FunctionElement`

`sage.interfaces.maxima.`**`is_MaximaElement`**(*x*)

Returns True if x is of type MaximaElement.

EXAMPLES:

```
sage: from sage.interfaces.maxima import is_MaximaElement
sage: m = maxima(1)
sage: is_MaximaElement(m)
True
sage: is_MaximaElement(1)
False
```

`sage.interfaces.maxima.`**`reduce_load_Maxima`**()

Unpickle a Maxima Pexpect interface.

EXAMPLES:

```
sage: from sage.interfaces.maxima import reduce_load_Maxima
sage: reduce_load_Maxima()
Maxima
```

`sage.interfaces.maxima.`**`reduce_load_Maxima_function`**(*parent*, *defn*, *args*, *latex*)

Unpickle a Maxima function.

EXAMPLES:

```
sage: from sage.interfaces.maxima import reduce_load_Maxima_function
sage: f = maxima.function('x,y','sin(x+y)')
sage: _,args = f.__reduce__()
sage: g = reduce_load_Maxima_function(*args)
sage: g == f
True
```

CHAPTER

FOURTEEN

LIBRARY INTERFACE TO MAXIMA

Maxima is a free GPL'd general purpose computer algebra system whose development started in 1968 at MIT. It contains symbolic manipulation algorithms, as well as implementations of special functions, including elliptic functions and generalized hypergeometric functions. Moreover, Maxima has implementations of many functions relating to the invariant theory of the symmetric group S_n. (However, the commands for group invariants, and the corresponding Maxima documentation, are in French.) For many links to Maxima documentation, see http://maxima.sourceforge.net/documentation.html.

AUTHORS:

- William Stein (2005-12): Initial version
- David Joyner: Improved documentation
- William Stein (2006-01-08): Fixed bug in parsing
- William Stein (2006-02-22): comparisons (following suggestion of David Joyner)
- William Stein (2006-02-24): *greatly* improved robustness by adding sequence numbers to IO bracketing in _eval_line
- Robert Bradshaw, Nils Bruin, Jean-Pierre Flori (2010,2011): Binary library interface

For this interface, Maxima is loaded into ECL which is itself loaded as a C library in Sage. Translations between Sage and Maxima objects (which are nothing but wrappers to ECL objects) is made as much as possible directly, but falls back to the string based conversion used by the classical Maxima Pexpect interface in case no new implementation has been made.

This interface is the one used for calculus by Sage and is accessible as $maxima_calculus$:

```
sage: maxima_calculus
Maxima_lib
```

Only one instance of this interface can be instantiated, so the user should not try to instantiate another one, which is anyway set to raise an error:

```
sage: from sage.interfaces.maxima_lib import MaximaLib
sage: MaximaLib()
Traceback (most recent call last):
...
RuntimeError: Maxima interface in library mode can only be instantiated once
```

class `sage.interfaces.maxima_lib.`**`MaximaLib`**

Bases: `sage.interfaces.maxima_abstract.MaximaAbstract`

Interface to Maxima as a Library.

INPUT: none

OUTPUT: Maxima interface as a Library

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import MaximaLib, maxima_lib
sage: isinstance(maxima_lib,MaximaLib)
True
```

Only one such interface can be instantiated:

```
sage: MaximaLib()
Traceback (most recent call last):
...
RuntimeError: Maxima interface in library mode can only
be instantiated once
```

clear(*var*)

Clear the variable named var.

INPUT:

- `var` - string

OUTPUT: none

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib.set('xxxxx', '2')
sage: maxima_lib.get('xxxxx')
'2'
sage: maxima_lib.clear('xxxxx')
sage: maxima_lib.get('xxxxx')
'xxxxx'
```

eval(*line*, *locals=None*, *reformat=True*, ***kwds*)

Evaluate the line in Maxima.

INPUT:

- `line` - string; text to evaluate
- `locals` - None (ignored); this is used for compatibility with the Sage notebook's generic system interface.
- `reformat` - boolean; whether to strip output or not
- `**kwds` - All other arguments are currently ignored.

OUTPUT: string representing Maxima output

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib._eval_line('1+1')
'2'
sage: maxima_lib._eval_line('1+1;')
'2'
sage: maxima_lib._eval_line('1+1$')
''
sage: maxima_lib._eval_line('randvar : cos(x)+sin(y)$')
''
```

```
sage: maxima_lib._eval_line('randvar')
'sin(y)+cos(x)'
```

get (*var*)

Get the string value of the variable `var`.

INPUT:

- `var` - string

OUTPUT: string

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib.set('xxxxx', '2')
sage: maxima_lib.get('xxxxx')
'2'
```

lisp (*cmd*)

Send a lisp command to maxima.

INPUT:

- `cmd` - string

OUTPUT: ECL object

Note: The output of this command is very raw - not pretty.

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib.lisp("(+ 2 17)")
<ECL: 19>
```

set (*var*, *value*)

Set the variable var to the given value.

INPUT:

- `var` - string
- `value` - string

OUTPUT: none

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib.set('xxxxx', '2')
sage: maxima_lib.get('xxxxx')
'2'
```

sr_integral (**args*)

Helper function to wrap calculus use of Maxima's integration.

TESTS:

```
sage: a,b=var('a,b')
sage: integrate(1/(x^3 *(a+b*x)^(1/3)),x)
Traceback (most recent call last):
...
```

```
ValueError: Computation failed since Maxima requested additional
constraints; using the 'assume' command before evaluation
*may* help (example of legal syntax is 'assume(a>0)', see
`assume?` for more details)
Is a positive or negative?
sage: assume(a>0)
sage: integrate(1/(x^3 *(a+b*x)^(1/3)),x)
2/9*sqrt(3)*b^2*arctan(1/3*sqrt(3)*(2*(b*x + a)^(1/3) + a^(1/3))/a^(1/3))/a^(7/3) - 1/9*b^2*
sage: var('x, n')
(x, n)
sage: integral(x^n,x)
Traceback (most recent call last):
...
ValueError: Computation failed since Maxima requested additional
constraints; using the 'assume' command before evaluation
*may* help (example of legal syntax is 'assume(n>0)',
see `assume?` for more details)
Is n equal to -1?
sage: assume(n+1>0)
sage: integral(x^n,x)
x^(n + 1)/(n + 1)
sage: forget()
sage: assumptions()  # Check the assumptions really were forgotten
[]
```

Make sure the abs_integrate package is being used, trac ticket #11483. The following are examples from the Maxima abs_integrate documentation:

```
sage: integrate(abs(x), x)
1/2*x*abs(x)

sage: integrate(sgn(x) - sgn(1-x), x)
abs(x - 1) + abs(x)

sage: integrate(1 / (1 + abs(x-5)), x, -5, 6)
log(11) + log(2)

sage: integrate(1/(1 + abs(x)), x)
1/2*(log(x + 1) + log(-x + 1))*sgn(x) + 1/2*log(x + 1) - 1/2*log(-x + 1)

sage: integrate(cos(x + abs(x)), x)
-1/4*(2*x - sin(2*x))*real_part(sgn(x)) + 1/2*x + 1/4*sin(2*x)
```

Note that the last example yielded the same answer in a simpler form in earlier versions of Maxima (<= 5.29.1), namely `-1/2*x*sgn(x) + 1/4*(sgn(x) + 1)*sin(2*x) + 1/2*x`. This is because Maxima no longer simplifies `realpart(signum(x))` to `signum(x)`:

```
sage: maxima("realpart(signum(x))")
'realpart(signum(x))
```

An example from sage-support thread e641001f8b8d1129:

```
sage: f = e^(-x^2/2)/sqrt(2*pi) * sgn(x-1)
sage: integrate(f, x, -Infinity, Infinity)
-erf(1/2*sqrt(2))
```

From trac ticket #8624:

```
sage: integral(abs(cos(x))*sin(x),(x,pi/2,pi))
1/2

sage: integrate(sqrt(x + sqrt(x)), x).simplify_radical()
1/12*((8*x - 3)*x^(1/4) + 2*x^(3/4))*sqrt(sqrt(x) + 1) + 1/8*log(sqrt(sqrt(x) + 1) + x^(1/4)
```

And trac ticket #11594:

```
sage: integrate(abs(x^2 - 1), x, -2, 2)
4
```

This definite integral returned zero (incorrectly) in at least Maxima 5.23. The correct answer is now given (trac ticket #11591):

```
sage: f = (x^2)*exp(x) / (1+exp(x))^2
sage: integrate(f, (x, -infinity, infinity))
1/3*pi^2
```

Sometimes one needs different simplification settings, such as `radexpand`, to compute an integral (see trac ticket #10955):

```
sage: f = sqrt(x + 1/x^2)
sage: maxima = sage.calculus.calculus.maxima
sage: maxima('radexpand')
true
sage: integrate(f, x)
integrate(sqrt(x + 1/x^2), x)
sage: maxima('radexpand: all')
all
sage: g = integrate(f, x); g
2/3*sqrt(x^3 + 1) - 1/3*log(sqrt(x^3 + 1) + 1) + 1/3*log(sqrt(x^3 + 1) - 1)
sage: (f - g.diff(x)).simplify_radical()
0
sage: maxima('radexpand: true')
true
```

The following integral was computed incorrectly in versions of Maxima before 5.27 (see trac ticket #12947):

```
sage: a = integrate(x*cos(x^3),(x,0,1/2)).n()
sage: a.real()
0.124756040961038
sage: a.imag().abs() < 3e-17
True
```

sr_limit (*expr*, *v*, *a*, *dir=None*)

Helper function to wrap calculus use of Maxima's limits.

TESTS:

```
sage: f = (1+1/x)^x
sage: limit(f,x = oo)
e
sage: limit(f,x = 5)
7776/3125
sage: limit(f,x = 1.2)
2.06961575467...
sage: var('a')
a
sage: limit(x^a,x=0)
```

```
Traceback (most recent call last):
...
ValueError: Computation failed since Maxima requested additional
constraints; using the 'assume' command before evaluation
*may* help (example of legal syntax is 'assume(a>0)', see `assume?`
for more details)
Is a positive, negative or zero?
sage: assume(a>0)
sage: limit(x^a,x=0)
Traceback (most recent call last):
...
ValueError: Computation failed ...
Is a an integer?
sage: assume(a,'integer')
sage: assume(a,'even')  # Yes, Maxima will ask this too
sage: limit(x^a,x=0)
0
sage: forget()
sage: assumptions() # check the assumptions were really forgotten
[]
```

The second limit below was computed incorrectly prior to Maxima 5.24 (trac ticket #10868):

```
sage: f(n) = 2 + 1/factorial(n)
sage: limit(f(n), n=infinity)
2
sage: limit(1/f(n), n=infinity)
1/2
```

The limit below was computed incorrectly prior to Maxima 5.30 (see trac ticket #13526):

```
sage: n = var('n')
sage: l = (3^n + (-2)^n) / (3^(n+1) + (-2)^(n+1))
sage: l.limit(n=oo)
1/3
```

sr_sum(**args*)

Helper function to wrap calculus use of Maxima's summation.

TESTS:

Check that trac ticket #16224 is fixed:

```
sage: k = var('k')
sage: sum(x^(2*k)/factorial(2*k), k, 0, oo).simplify_radical()
cosh(x)

sage: x, y, k, n = var('x, y, k, n')
sage: sum(binomial(n,k) * x^k * y^(n-k), k, 0, n)
(x + y)^n
sage: q, a = var('q, a')
sage: sum(a*q^k, k, 0, oo)
Traceback (most recent call last):
...
ValueError: Computation failed since Maxima requested additional
constraints; using the 'assume' command before evaluation *may* help
(example of legal syntax is 'assume(abs(q)-1>0)', see `assume?`
for more details)
Is abs(q)-1 positive, negative or zero?
sage: assume(q > 1)
```

```
sage: sum(a*q^k, k, 0, oo)
Traceback (most recent call last):
...
ValueError: Sum is divergent.
sage: forget()
sage: assume(abs(q) < 1)
sage: sum(a*q^k, k, 0, oo)
-a/(q - 1)
sage: forget()
sage: assumptions() # check the assumptions were really forgotten
[]
```

Taking the sum of all natural numbers informs us that the sum is divergent. Maxima (before 5.29.1) used to ask questions about m, leading to a different error (see trac ticket #11990):

```
sage: m = var('m')
sage: sum(m, m, 0, infinity)
Traceback (most recent call last):
...
ValueError: Sum is divergent.
```

An error with an infinite sum in Maxima (before 5.30.0, see trac ticket #13712):

```
sage: n = var('n')
sage: sum(1/((2*n-1)^2*(2*n+1)^2*(2*n+3)^2), n, 0, oo)
3/256*pi^2
```

Maxima correctly detects division by zero in a symbolic sum (see trac ticket #11894):

```
sage: sum(1/(m^4 + 2*m^3 + 3*m^2 + 2*m)^2, m, 0, infinity)
Traceback (most recent call last):
...
RuntimeError: ECL says: Error executing code in Maxima: Zero to negative power computed.
```

Similar situation for trac ticket #12410:

```
sage: x = var('x')
sage: sum(1/x*(-1)^x, x, 0, oo)
Traceback (most recent call last):
...
RuntimeError: ECL says: Error executing code in Maxima: Zero to negative power computed.
```

sr_tlimit (*expr*, *v*, *a*, *dir=None*)

Helper function to wrap calculus use of Maxima's Taylor series limits.

TESTS:

```
sage: f = (1+1/x)^x
sage: limit(f, x = I, taylor=True)
(-I + 1)^I
```

class sage.interfaces.maxima_lib.**MaximaLibElement** (*parent*, *value*, *is_name=False*, *name=None*)

Bases: sage.interfaces.maxima_abstract.MaximaAbstractElement

Element of Maxima through library interface.

EXAMPLES:

Elements of this class should not be created directly. The targeted parent should be used instead:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib(4)
4
sage: maxima_lib(log(x))
log(_SAGE_VAR_x)
```

display2d(*onscreen=True*)

Return the 2d representation of this Maxima object.

INPUT:

- `onscreen` - boolean (default: True); whether to print or return

OUTPUT:

The representation is printed if onscreen is set to True and returned as a string otherwise.

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: F = maxima_lib('x^5 - y^5').factor()
sage: F.display2d()
                           4      3    2  2    3      4
               - (y - x) (y  + x y  + x  y  + x  y + x )
```

ecl()

Return the underlying ECL object of this MaximaLib object.

INPUT: none

OUTPUT: ECL object

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib(x+cos(19)).ecl()
<ECL: ((MPLUS SIMP) ((%COS SIMP) 19) |$_SAGE_VAR_x|)>
```

to_poly_solve(*vars, options=''*)

Use Maxima's to_poly_solver package.

INPUT:

- `vars` - symbolic expressions
- `options` - string (default="")

OUTPUT: Maxima object

EXAMPLES:

The zXXX below are names for arbitrary integers and subject to change:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: sol = maxima_lib(sin(x) == 0).to_poly_solve(x)
sage: sol.sage()
[[x == pi*z54]]
```

class sage.interfaces.maxima_lib.**MaximaLibElementFunction**(*parent, name, defn, args, latex*)

Bases: sage.interfaces.maxima_lib.MaximaLibElement, sage.interfaces.maxima_abstract.Maxima

Create a Maxima function. See `MaximaAbstractElementFunction` for full documentation.

TESTS:

```
sage: from sage.interfaces.maxima_abstract import MaximaAbstractElementFunction
sage: MaximaAbstractElementFunction == loads(dumps(MaximaAbstractElementFunction))
True
sage: f = maxima.function('x,y','sin(x+y)')
sage: f == loads(dumps(f))
True
```

class `sage.interfaces.maxima_lib.`**`MaximaLibFunction`**(*parent*, *name*)

Bases: `sage.interfaces.maxima_abstract.MaximaAbstractFunction`

class `sage.interfaces.maxima_lib.`**`MaximaLibFunctionElement`**(*obj*, *name*)

Bases: `sage.interfaces.maxima_abstract.MaximaAbstractFunctionElement`

`sage.interfaces.maxima_lib.`**`add_vararg`**(**args*)

Addition of a variable number of arguments.

INPUT:

- `args` - arguments to add

OUTPUT: sum of arguments

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import add_vararg
sage: add_vararg(1,2,3,4,5,6,7)
28
```

`sage.interfaces.maxima_lib.`**`dummy_integrate`**(*expr*)

We would like to simply tie Maxima's integrate to sage.calculus.calculus.dummy_integrate, but we're being imported there so to avoid circularity we define it here.

INPUT:

- `expr` - ECL object; a Maxima %INTEGRATE expression

OUTPUT: symbolic expression

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, dummy_integrate
sage: f = maxima_lib('f(x)').integrate('x')
sage: f.ecl()
<ECL: ((%INTEGRATE SIMP) (($F SIMP) $X) $X)>
sage: dummy_integrate(f.ecl())
integrate(f(x), x)

sage: f = maxima_lib('f(x)').integrate('x',0,10)
sage: f.ecl()
<ECL: ((%INTEGRATE SIMP) (($F SIMP) $X) $X 0 10)>
sage: dummy_integrate(f.ecl())
integrate(f(x), x, 0, 10)
```

`sage.interfaces.maxima_lib.`**`is_MaximaLibElement`**(*x*)

Returns True if x is of type MaximaLibElement.

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, is_MaximaLibElement
sage: m = maxima_lib(1)
sage: is_MaximaLibElement(m)
True
```

```
sage: is_MaximaLibElement(1)
False
```

sage.interfaces.maxima_lib.**max_at_to_sage**(*expr*)

Special conversion rule for AT expressions.

INPUT:

- `expr` - ECL object; a Maxima AT expression

OUTPUT: symbolic expression

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, max_at_to_sage
sage: a=maxima_lib("'at(f(x,y,z),[x=1,y=2,z=3])")
sage: a
'at(f(x,y,z),[x=1,y=2,z=3])
sage: max_at_to_sage(a.ecl())
f(1, 2, 3)
sage: a=maxima_lib("'at(f(x,y,z),x=1)")
sage: a
'at(f(x,y,z),x=1)
sage: max_at_to_sage(a.ecl())
f(1, y, z)
```

sage.interfaces.maxima_lib.**max_to_sr**(*expr*)

Convert a Maxima object into a symbolic expression.

INPUT:

- `expr` - ECL object

OUTPUT: symbolic expression

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, max_to_sr
sage: f = maxima_lib('f(x)')
sage: f.ecl()
<ECL: (($F SIMP) $X)>
sage: max_to_sr(f.ecl())
f(x)
```

TESTS:

```
sage: from sage.interfaces.maxima_lib import sr_to_max, max_to_sr
sage: f = function('f',x).diff()
sage: bool(max_to_sr(sr_to_max(f)) == f)
True
```

sage.interfaces.maxima_lib.**max_to_string**(*s*)

Return the Maxima string corresponding to this ECL object.

INPUT:

- `s` - ECL object

OUTPUT: string

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, max_to_string
sage: ecl = maxima_lib(cos(x)).ecl()
sage: max_to_string(ecl)
'cos(_SAGE_VAR_x)'
```

sage.interfaces.maxima_lib.**mdiff_to_sage**(*expr*)

Special conversion rule for %DERIVATIVE expressions.

INPUT:

- expr - ECL object; a Maxima %DERIVATIVE expression

OUTPUT: symbolic expression

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, mdiff_to_sage
sage: f = maxima_lib('f(x)').diff('x',4)
sage: f.ecl()
<ECL: ((%DERIVATIVE SIMP) (($F SIMP) $X) $X 4)>
sage: mdiff_to_sage(f.ecl())
D[0, 0, 0, 0](f)(x)
```

sage.interfaces.maxima_lib.**mlist_to_sage**(*expr*)

Special conversion rule for MLIST expressions.

INPUT:

- expr - ECL object; a Maxima MLIST expression (i.e., a list)

OUTPUT: a Python list of converted expressions.

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, mlist_to_sage
sage: L=maxima_lib("[1,2,3]")
sage: L.ecl()
<ECL: ((MLIST SIMP) 1 2 3)>
sage: mlist_to_sage(L.ecl())
[1, 2, 3]
```

sage.interfaces.maxima_lib.**mqapply_to_sage**(*expr*)

Special conversion rule for MQAPPLY expressions.

INPUT:

- expr - ECL object; a Maxima MQAPPLY expression

OUTPUT: symbolic expression

MQAPPLY is used for function as li[x](y) and psi[x](y).

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, mqapply_to_sage
sage: c = maxima_lib('li[2](3)')
sage: c.ecl()
<ECL: ((MQAPPLY SIMP) (($LI SIMP ARRAY) 2) 3)>
sage: mqapply_to_sage(c.ecl())
polylog(2, 3)
```

sage.interfaces.maxima_lib.**mrat_to_sage**(*expr*)

Convert a Maxima MRAT expression to Sage SR.

INPUT:

- `expr` - ECL object; a Maxima MRAT expression

OUTPUT: symbolic expression

Maxima has an optimised representation for multivariate rational expressions. The easiest way to translate those to SR is by first asking Maxima to give the generic representation of the object. That is what RATDISREP does in Maxima.

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, mrat_to_sage
sage: var('x y z')
(x, y, z)
sage: c = maxima_lib((x+y^2+z^9)/x^6+z^8/y).rat()
sage: c
(_SAGE_VAR_y*_SAGE_VAR_z^9+_SAGE_VAR_x^6*_SAGE_VAR_z^8+_SAGE_VAR_y^3+_SAGE_VAR_x*_SAGE_VAR_y)/(_
sage: c.ecl()
<ECL: ((MRAT SIMP (|$_SAGE_VAR_x| |$_SAGE_VAR_y| |$_SAGE_VAR_z|)
...>
sage: mrat_to_sage(c.ecl())
(x^6*z^8 + y*z^9 + y^3 + x*y)/(x^6*y)
```

`sage.interfaces.maxima_lib.`**`mul_vararg`**`(*args)`

Multiplication of a variable number of arguments.

INPUT:

- `args` - arguments to multiply

OUTPUT: product of arguments

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import mul_vararg
sage: mul_vararg(9,8,7,6,5,4)
60480
```

`sage.interfaces.maxima_lib.`**`parse_max_string`**`(s)`

Evaluate string in Maxima without *any* further simplification.

INPUT:

- `s` - string

OUTPUT: ECL object

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import parse_max_string
sage: parse_max_string('1+1')
<ECL: ((MPLUS) 1 1)>
```

`sage.interfaces.maxima_lib.`**`pyobject_to_max`**`(obj)`

Convert a (simple) Python object into a Maxima object.

INPUT:

- `expr` - Python object

OUTPUT: ECL object

Note: This uses functions defined in sage.libs.ecl.

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import pyobject_to_max
sage: pyobject_to_max(4)
<ECL: 4>
sage: pyobject_to_max('z')
<ECL: Z>
sage: var('x')
x
sage: pyobject_to_max(x)
Traceback (most recent call last):
...
TypeError: Unimplemented type for python_to_ecl
```

`sage.interfaces.maxima_lib.`**`reduce_load_MaximaLib`**`()`

Unpickle the (unique) Maxima library interface.

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import reduce_load_MaximaLib
sage: reduce_load_MaximaLib()
Maxima_lib
```

`sage.interfaces.maxima_lib.`**`sage_rat`**`(`*x, y*`)`

Return quotient x/y.

INPUT:

- `x` - integer
- `y` - integer

OUTPUT: rational

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import sage_rat
sage: sage_rat(1,7)
1/7
```

`sage.interfaces.maxima_lib.`**`sr_to_max`**`(`*expr*`)`

Convert a symbolic expression into a Maxima object.

INPUT:

- `expr` - symbolic expression

OUTPUT: ECL object

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import sr_to_max
sage: var('x')
x
sage: sr_to_max(x)
<ECL: $X>
sage: sr_to_max(cos(x))
<ECL: ((%COS) $X)>
sage: f = function('f',x)
sage: sr_to_max(f.diff())
<ECL: ((%DERIVATIVE) (($F) $X) $X 1)>
```

TESTS:

We should be able to convert derivatives evaluated at a point, trac ticket #12796:

```
sage: from sage.interfaces.maxima_lib import sr_to_max, max_to_sr
sage: f = function('f')
sage: f_prime = f(x).diff(x)
sage: max_to_sr(sr_to_max(f_prime(x = 1)))
D[0](f)(1)
```

`sage.interfaces.maxima_lib.`**`stdout_to_string`**`(s)`

Evaluate command `s` and catch Maxima stdout (not the result of the command!) into a string.

INPUT:

- `s` - string; command to evaluate

OUTPUT: string

This is currently used to implement `display2d()`.

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import stdout_to_string
sage: stdout_to_string('1+1')
''
sage: stdout_to_string('disp(1+1)')
'2\n\n'
```

CHAPTER

FIFTEEN

INTERFACE TO MATHEMATICA

The Mathematica interface will only work if Mathematica is installed on your computer with a command line interface that runs when you give the `math` command. The interface lets you send certain Sage objects to Mathematica, run Mathematica functions, import certain Mathematica expressions to Sage, or any combination of the above.

To send a Sage object `sobj` to Mathematica, call `mathematica(sobj)`. This exports the Sage object to Mathematica and returns a new Sage object wrapping the Mathematica expression/variable, so that you can use the Mathematica variable from within Sage. You can then call Mathematica functions on the new object; for example:

```
sage: mobj = mathematica(x^2-1)              # optional - mathematica
sage: mobj.Factor()                          # optional - mathematica
(-1 + x)*(1 + x)
```

In the above example the factorization is done using Mathematica's `Factor[]` function.

To see Mathematica's output you can simply print the Mathematica wrapper object. However if you want to import Mathematica's output back to Sage, call the Mathematica wrapper object's `sage()` method. This method returns a native Sage object:

```
sage: mobj = mathematica(x^2-1)              # optional - mathematica
sage: mobj2 = mobj.Factor(); mobj2           # optional - mathematica
(-1 + x)*(1 + x)
sage: mobj2.parent()                         # optional - mathematica
Mathematica
sage: sobj = mobj2.sage(); sobj              # optional - mathematica
(x - 1)*(x + 1)
sage: sobj.parent()                          # optional - mathematica
Symbolic Ring
```

If you want to run a Mathematica function and don't already have the input in the form of a Sage object, then it might be simpler to input a string to `mathematica(expr)`. This string will be evaluated as if you had typed it into Mathematica:

```
sage: mathematica('Factor[x^2-1]')           # optional - mathematica
(-1 + x)*(1 + x)
sage: mathematica('Range[3]')                # optional - mathematica
{1, 2, 3}
```

If you don't want Sage to go to the trouble of creating a wrapper for the Mathematica expression, then you can call `mathematica.eval(expr)`, which returns the result as a Mathematica AsciiArtString formatted string. If you want the result to be a string formatted like Mathematica's InputForm, call `repr(mobj)` on the wrapper object `mobj`. If you want a string formatted in Sage style, call `mobj._sage_repr()`:

```
sage: mathematica.eval('x^2 - 1')            # optional - mathematica
                2
          -1 + x
sage: repr(mathematica('Range[3]'))          # optional - mathematica
'{1, 2, 3}'
sage: mathematica('Range[3]')._sage_repr()  # optional - mathematica
'[1, 2, 3]'
```

Finally, if you just want to use a Mathematica command line from within Sage, the function `mathematica_console()` dumps you into an interactive command-line Mathematica session. This is an enhanced version of the usual Mathematica command-line, in that it provides readline editing and history (the usual one doesn't!)

15.1 Tutorial

We follow some of the tutorial from http://library.wolfram.com/conferences/devconf99/withoff/Basic1.html/.

For any of this to work you must buy and install the Mathematica program, and it must be available as the command `math` in your PATH.

15.1.1 Syntax

Now make 1 and add it to itself. The result is a Mathematica object.

```
sage: m = mathematica
sage: a = m(1) + m(1); a                  # optional - mathematica
2
sage: a.parent()                          # optional - mathematica
Mathematica
sage: m('1+1')                            # optional - mathematica
2
sage: m(3)**m(50)                         # optional - mathematica
717897987691852588770249
```

The following is equivalent to `Plus[2, 3]` in Mathematica:

```
sage: m = mathematica
sage: m(2).Plus(m(3))                     # optional - mathematica
5
```

We can also compute $7(2+3)$.

```
sage: m(7).Times(m(2).Plus(m(3)))         # optional - mathematica
35
sage: m('7(2+3)')                         # optional - mathematica
35
```

15.1.2 Some typical input

We solve an equation and a system of two equations:

```
sage: eqn = mathematica('3x + 5 == 14') # optional - mathematica
sage: eqn                               # optional - mathematica
5 + 3*x == 14
sage: eqn.Solve('x')                    # optional - mathematica
{{x -> 3}}
sage: sys = mathematica('{x^2 - 3y == 3, 2x - y == 1}')  # optional - mathematica
sage: print sys                         # optional - mathematica
           2
         {x  - 3 y == 3, 2 x - y == 1}
sage: sys.Solve('{x, y}')               # optional - mathematica
{{x -> 0, y -> -1}, {x -> 6, y -> 11}}
```

15.1.3 Assignments and definitions

If you assign the mathematica 5 to a variable c in Sage, this does not affect the c in Mathematica.

```
sage: c = m(5)                          # optional - mathematica
sage: print m('b + c x')                # optional - mathematica
             b + c x
sage: print m('b') + c*m('x')           # optional - mathematica
         b + 5 x
```

The Sage interfaces changes Sage lists into Mathematica lists:

```
sage: m = mathematica
sage: eq1 = m('x^2 - 3y == 3')          # optional - mathematica
sage: eq2 = m('2x - y == 1')            # optional - mathematica
sage: v = m([eq1, eq2]); v              # optional - mathematica
{x^2 - 3*y == 3, 2*x - y == 1}
sage: v.Solve(['x', 'y'])               # optional - mathematica
{{x -> 0, y -> -1}, {x -> 6, y -> 11}}
```

15.1.4 Function definitions

Define mathematica functions by simply sending the definition to the interpreter.

```
sage: m = mathematica
sage: _ = mathematica('f[p_] = p^2');    # optional - mathematica
sage: m('f[9]')                         # optional - mathematica
81
```

15.1.5 Numerical Calculations

We find the x such that $e^x - 3x = 0$.

```
sage: e = mathematica('Exp[x] - 3x == 0') # optional - mathematica
sage: e.FindRoot(['x', 2])                # optional - mathematica
{x -> 1.512134551657842}
```

Note that this agrees with what the PARI interpreter gp produces:

```
sage: gp('solve(x=1,2,exp(x)-3*x)')
1.512134551657842473896739678             # 32-bit
1.5121345516578424738967396780720387046    # 64-bit
```

Next we find the minimum of a polynomial using the two different ways of accessing Mathematica:

```
sage: mathematica('FindMinimum[x^3 - 6x^2 + 11x - 5, {x,3}]')  # optional - mathematica
{0.6150998205402516, {x -> 2.5773502699629733}}
sage: f = mathematica('x^3 - 6x^2 + 11x - 5')  # optional - mathematica
sage: f.FindMinimum(['x', 3])                  # optional - mathematica
{0.6150998205402516, {x -> 2.5773502699629733}}
```

15.1.6 Polynomial and Integer Factorization

We factor a polynomial of degree 200 over the integers.

```
sage: R.<x> = PolynomialRing(ZZ)
sage: f = (x**100+17*x+5)*(x**100-5*x+20)
sage: f
x^200 + 12*x^101 + 25*x^100 - 85*x^2 + 315*x + 100
sage: g = mathematica(str(f))            # optional - mathematica
sage: print g                            # optional - mathematica
                           2       100       101    200
         100 + 315 x - 85 x  + 25 x    + 12 x    + x
sage: g                                  # optional - mathematica
100 + 315*x - 85*x^2 + 25*x^100 + 12*x^101 + x^200
sage: print g.Factor()                   # optional - mathematica
                      100             100
         (20 - 5 x + x   ) (5 + 17 x + x   )
```

We can also factor a multivariate polynomial:

```
sage: f = mathematica('x^6 + (-y - 2)*x^5 + (y^3 + 2*y)*x^4 - y^4*x^3')  # optional - mathematica
sage: print f.Factor()                   # optional - mathematica
          3                  2    3
         x  (x - y) (-2 x + x  + y )
```

We factor an integer:

```
sage: n = mathematica(2434500)           # optional - mathematica
sage: n.FactorInteger()                  # optional - mathematica
{{2, 2}, {3, 2}, {5, 3}, {541, 1}}
sage: n = mathematica(2434500)           # optional - mathematica
sage: F = n.FactorInteger(); F           # optional - mathematica
{{2, 2}, {3, 2}, {5, 3}, {541, 1}}
sage: F[1]                               # optional - mathematica
{2, 2}
sage: F[4]                               # optional - mathematica
{541, 1}
```

We can also load the ECM package and factoring using it:

```
sage: _ = mathematica.eval("<<NumberTheory`FactorIntegerECM`");  # optional - mathematica
sage: mathematica.FactorIntegerECM('932901*939321')              # optional - mathematica
8396109
```

15.2 Long Input

The Mathematica interface reads in even very long input (using files) in a robust manner.

```
sage: t = '"%s"'%10^10000   # ten thousand character string.
sage: a = mathematica(t)        # optional - mathematica
sage: a = mathematica.eval(t)   # optional - mathematica
```

15.3 Loading and saving

Mathematica has an excellent `InputForm` function, which makes saving and loading Mathematica objects possible. The first examples test saving and loading to strings.

```
sage: x = mathematica(pi/2)     # optional - mathematica
sage: print x                   # optional - mathematica
        Pi
        --
        2
sage: loads(dumps(x)) == x      # optional - mathematica
True
sage: n = x.N(50)               # optional - mathematica
sage: print n                   # optional - mathematica
           1.5707963267948966192313216916397514420985846996876
sage: loads(dumps(n)) == n      # optional - mathematica
True
```

15.4 Complicated translations

The `mobj.sage()` method tries to convert a Mathematica object to a Sage object. In many cases, it will just work. In particular, it should be able to convert expressions entirely consisting of:

- numbers, i.e. integers, floats, complex numbers;
- functions and named constants also present in Sage, where:
  - Sage knows how to translate the function or constant's name from Mathematica's, or
  - the Sage name for the function or constant is trivially related to Mathematica's;
- symbolic variables whose names don't pathologically overlap with objects already defined in Sage.

This method will not work when Mathematica's output includes:

- strings;
- functions unknown to Sage;
- Mathematica functions with different parameters/parameter order to the Sage equivalent.

If you want to convert more complicated Mathematica expressions, you can instead call `mobj._sage_()` and supply a translation dictionary:

```
sage: m = mathematica('NewFn[x]')           # optional - mathematica
sage: m._sage_(locals={'NewFn': sin})    # optional - mathematica
sin(x)
```

For more details, see the documentation for `._sage_()`.

OTHER Examples:

```
sage: def math_bessel_K(nu,x):
...       return mathematica(nu).BesselK(x).N(20)
...
sage: math_bessel_K(2,I)                      # optional - mathematica
0.180489972066962*I - 2.592886175491197             # 32-bit
-2.59288617549119697817 + 0.18048997206696202663*I  # 64-bit
```

```
sage: slist = [[1, 2], 3., 4 + I]
sage: mlist = mathematica(slist); mlist     # optional - mathematica
{{1, 2}, 3., 4 + I}
sage: slist2 = list(mlist); slist2          # optional - mathematica
[{1, 2}, 3., 4 + I]
sage: slist2[0]                             # optional - mathematica
{1, 2}
sage: slist2[0].parent()                    # optional - mathematica
Mathematica
sage: slist3 = mlist.sage(); slist3         # optional - mathematica
[[1, 2], 3.0, I + 4]
```

```
sage: mathematica('10.^80')      # optional - mathematica
1.*^80
sage: mathematica('10.^80').sage()   # optional - mathematica
1e+80
```

AUTHORS:

- William Stein (2005): first version
- Doug Cutrell (2006-03-01): Instructions for use under Cygwin/Windows.
- Felix Lawrence (2009-08-21): Added support for importing Mathematica lists and floats with exponents.

class `sage.interfaces.mathematica.`**`Mathematica`**(*maxread=100*, *script_subdirectory=''*, *logfile=None*, *server=None*, *server_tmpdir=None*)

Bases: `sage.interfaces.expect.Expect`

Interface to the Mathematica interpreter.

`chdir`(*dir*)

Change Mathematica's current working directory.

EXAMPLES:

```
sage: mathematica.chdir('/')           # optional
sage: mathematica('Directory[]')       # optional
"/"
```

`console`(*readline=True*)

x.__init__(...) initializes x; see help(type(x)) for signature

`eval`(*code*, *strip=True*, ***kwds*)

x.__init__(...) initializes x; see help(type(x)) for signature

`get`(*var*, *ascii_art=False*)

Get the value of the variable var.

AUTHORS:

- William Stein
- Kiran Kedlaya (2006-02-04): suggested using InputForm

help (*cmd*)
x.__init__(...) initializes x; see help(type(x)) for signature

set (*var*, *value*)
Set the variable var to the given value.

trait_names ()
x.__init__(...) initializes x; see help(type(x)) for signature

class sage.interfaces.mathematica.**MathematicaElement** (*parent*, *value*, *is_name=False*, *name=None*)
Bases: sage.interfaces.expect.ExpectElement

N (**args*)
EXAMPLES:

```
sage: mathematica('Pi').N(10)    # optional -- mathematica
3.1415926536
sage: mathematica('Pi').N(50)    # optional -- mathematica
3.1415926535897932384626433832795028841971693993751
```

show (*filename=None*, *ImageSize=600*)
Show a mathematica expression or plot in the Sage notebook.

EXAMPLES:

```
sage: P = mathematica('Plot[Sin[x],{x,-2Pi,4Pi}]')   # optional - mathematica
sage: show(P)                                          # optional - mathematica
sage: P.show(ImageSize=800)                            # optional - mathematica
sage: Q = mathematica('Sin[x Cos[y]]/Sqrt[1-x^2]')    # optional - mathematica
sage: show(Q)                                          # optional - mathematica
<html><div class="math">\frac{\sin (x \cos (y))}{\sqrt{1-x^2}}</div></html>
```

str ()
x.__init__(...) initializes x; see help(type(x)) for signature

class sage.interfaces.mathematica.**MathematicaFunction** (*parent*, *name*)
Bases: sage.interfaces.expect.ExpectFunction

class sage.interfaces.mathematica.**MathematicaFunctionElement** (*obj*, *name*)
Bases: sage.interfaces.expect.FunctionElement

sage.interfaces.mathematica.**clean_output** (*s*)
x.__init__(...) initializes x; see help(type(x)) for signature

sage.interfaces.mathematica.**mathematica_console** (*readline=True*)
x.__init__(...) initializes x; see help(type(x)) for signature

sage.interfaces.mathematica.**reduce_load** (*X*)
x.__init__(...) initializes x; see help(type(x)) for signature

CHAPTER

SIXTEEN

INTERFACE TO MWRANK

sage.interfaces.mwrank.**Mwrank**(*options=''*, *server=None*, *server_tmpdir=None*)
Create and return an mwrank interpreter, with given options.

INPUT:

- `options` - string; passed when starting mwrank. The format is:

```
-h       help            prints this info and quits
-q       quiet           turns OFF banner display and prompt
-v n     verbosity       sets verbosity to n (default=1)
-o       PARI/GP output  turns ON extra PARI/GP short output (default is OFF)
-p n     precision       sets precision to n decimals (default=15)
-b n     quartic bound   bound on quartic point search (default=10)
-x n     n aux           number of aux primes used for sieving (default=6)
-l       list            turns ON listing of points (default ON unless v=0)
-s       selmer_only     if set, computes Selmer rank only (default: not set)
-d       skip_2nd_descent        if set, skips the second descent for curves with 2-torsion
-S n     sat_bd          upper bound on saturation primes (default=100, -1 for automatic)
```

EXAMPLES:

```
sage: M = Mwrank('-v 0 -l')
sage: print M('0 0 1 -1 0')
Curve [0,0,1,-1,0] :    Rank = 1
Generator 1 is [0:-1:1]; height 0.0511114082399688
Regulator = 0.0511114082399688
```

class sage.interfaces.mwrank.**Mwrank_class**(*options=''*, *server=None*, *server_tmpdir=None*)
Bases: `sage.interfaces.expect.Expect`

Interface to the Mwrank interpreter.

console()
Start the mwrank console.

EXAMPLE:

```
sage: mwrank.console() # not tested: expects console input
Program mwrank: ...
```

eval(*s*, ***kwds*)
Return mwrank's output for the given input.

INPUT:

- `s` (str) - a Sage object which when converted to a string gives valid input to `mwrank`. The conversion is done by `validate_mwrank_input()`. Possible formats are:
  - a string representing exactly five integers separated by whitespace, for example '1 2 3 4 5'
  - a string representing exactly five integers separated by commas, preceded by '[' and followed by ']' (with arbitrary whitespace), for example '[1 2 3 4 5]'
  - a list or tuple of exactly 5 integers.

Note: If a RuntimeError exception is raised, then the mwrank interface is restarted and the command is retried once.

EXAMPLES:

```
sage: mwrank.eval('12 3 4 5 6')
'Curve [12,3,4,5,6] :...'
sage: mwrank.eval('[12, 3, 4, 5, 6]')
'Curve [12,3,4,5,6] :...'
sage: mwrank.eval([12, 3, 4, 5, 6])
'Curve [12,3,4,5,6] :...'
sage: mwrank.eval((12, 3, 4, 5, 6))
'Curve [12,3,4,5,6] :...'
```

quit(*verbose=False*)

Quit the mwrank process using kill -9 (so exit doesn't dump core, etc.).

INPUT:

- `verbose` – ignored

EXAMPLES:

```
sage: m = Mwrank()
sage: e = m('1 2 3 4 5')
sage: m.quit()
```

sage.interfaces.mwrank.**mwrank_console**()

Start the mwrank console.

EXAMPLE:

```
sage: mwrank_console() # not tested: expects console input
Program mwrank: ...
```

sage.interfaces.mwrank.**validate_mwrank_input**(*s*)

Returns a string suitable for mwrank input, or raises an error.

INPUT:

- s – one of the following:
  - a list or tuple of 5 integers [a1,a2,a3,a4,a6] or (a1,a2,a3,a4,a6)
  - a string of the form '[a1,a2,a3,a4,a6]' or 'a1 a2 a3 a4 a6' where a1, a2, a3, a4, a6 are integers

OUTPUT:

For valid input, a string of the form '[a1,a2,a3,a4,a6]'. For invalid input a ValueError is raised.

EXAMPLES:

A list or tuple of 5 integers:

```
sage: from sage.interfaces.mwrank import validate_mwrank_input
sage: validate_mwrank_input([1,2,3,4,5])
'[1, 2, 3, 4, 5]'
sage: validate_mwrank_input((-1,2,-3,4,-55))
'[-1, 2, -3, 4, -55]'
sage: validate_mwrank_input([1,2,3,4])
Traceback (most recent call last):
...
ValueError: [1, 2, 3, 4] is not valid input to mwrank (should have 5 entries)
sage: validate_mwrank_input([1,2,3,4,i])
Traceback (most recent call last):
...
ValueError: [1, 2, 3, 4, I] is not valid input to mwrank (entries should be integers)
```

A string of the form '[a1,a2,a3,a4,a6]' with any whitespace and integers ai:

```
sage: validate_mwrank_input('0 -1 1 -7 6')
'[0,-1,1,-7,6]'
sage: validate_mwrank_input("[0,-1,1,0,0]\n")
'[0,-1,1,0,0]'
sage: validate_mwrank_input('0\t -1\t 1\t 0\t 0\n')
'[0,-1,1,0,0]'
sage: validate_mwrank_input('0 -1 1 -7 ')
Traceback (most recent call last):
...
ValueError: 0 -1 1 -7  is not valid input to mwrank
```

CHAPTER

SEVENTEEN

INTERFACE TO GNU OCTAVE

GNU Octave is a free software (GPL) MATLAB-like program with numerical routines for integrating, solving systems of equations, special functions, and solving (numerically) differential equations. Please see http://octave.org/ for more details.

The commands in this section only work if you have the optional "octave" interpreter installed and available in your PATH. It's not necessary to install any special Sage packages.

EXAMPLES:

```
sage: octave.eval('2+2')     # optional - octave
'ans = 4'

sage: a = octave(10)         # optional - octave
sage: a**10                  # optional - octave
1e+10
```

LOG: - creation (William Stein) - ? (David Joyner, 2005-12-18) - Examples (David Joyner, 2005-01-03)

17.1 Computation of Special Functions

Octave implements computation of the following special functions (see the maxima and gp interfaces for even more special functions):

```
airy
    Airy functions of the first and second kind, and their derivatives.
    airy(0,x) = Ai(x), airy(1,x) = Ai'(x), airy(2,x) = Bi(x), airy(3,x) = Bi'(x)
besselj
    Bessel functions of the first kind.
bessely
    Bessel functions of the second kind.
besseli
    Modified Bessel functions of the first kind.
besselk
    Modified Bessel functions of the second kind.
besselh
    Compute Hankel functions of the first (k = 1) or second (k = 2) kind.
beta
    The Beta function,
          beta (a, b) = gamma (a) * gamma (b) / gamma (a + b).
betainc
    The incomplete Beta function,
erf
```

```
    The error function,
erfinv
    The inverse of the error function.
gamma
    The Gamma function,
gammainc
    The incomplete gamma function,
```

For example,

```
sage: octave("airy(3,2)")         # optional - octave
4.10068
sage: octave("beta(2,2)")         # optional - octave
0.166667
sage: octave("betainc(0.2,2,2)")  # optional - octave
0.104
sage: octave("besselh(0,2)")      # optional - octave
(0.223891,0.510376)
sage: octave("besselh(0,1)")      # optional - octave
(0.765198,0.088257)
sage: octave("besseli(1,2)")      # optional - octave
1.59064
sage: octave("besselj(1,2)")      # optional - octave
0.576725
sage: octave("besselk(1,2)")      # optional - octave
0.139866
sage: octave("erf(0)")            # optional - octave
0
sage: octave("erf(1)")            # optional - octave
0.842701
sage: octave("erfinv(0.842)")     # optional - octave
0.998315
sage: octave("gamma(1.5)")        # optional - octave
0.886227
sage: octave("gammainc(1.5,1)")   # optional - octave
0.77687
```

The Octave interface reads in even very long input (using files) in a robust manner:

```
sage: t = '"%s"'%10^10000   # ten thousand character string.
sage: a = octave.eval(t + ';')    # optional - octave, < 1/100th of a second
sage: a = octave(t)               # optional - octave
```

Note that actually reading a back out takes forever. This *must* be fixed ASAP - see http://trac.sagemath.org/sage_trac/ticket/940/.

17.2 Tutorial

EXAMPLES:

```
sage: octave('4+10')              # optional - octave
14
sage: octave('date')              # optional - octave; random output
18-Oct-2007
sage: octave('5*10 + 6')          # optional - octave
56
```

```
sage: octave('(6+6)/3')           # optional - octave
4
sage: octave('9')^2               # optional - octave
81
sage: a = octave(10); b = octave(20); c = octave(30)     # optional - octave
sage: avg = (a+b+c)/3             # optional - octave
sage: avg                         # optional - octave
20
sage: parent(avg)                 # optional - octave
Octave
```

```
sage: my_scalar = octave('3.1415')        # optional - octave
sage: my_scalar                           # optional - octave
3.1415
sage: my_vector1 = octave('[1,5,7]')      # optional - octave
sage: my_vector1                          # optional - octave
1     5     7
sage: my_vector2 = octave('[1;5;7]')      # optional - octave
sage: my_vector2                          # optional - octave
1
5
7
sage: my_vector1 * my_vector2             # optional - octave
75
```

class `sage.interfaces.octave.`**`Octave`**(*maxread=100*, *script_subdirectory=''*, *logfile=None*, *server=None*, *server_tmpdir=None*)

Bases: `sage.interfaces.expect.Expect`

Interface to the Octave interpreter.

EXAMPLES:

```
sage: octave.eval("a = [ 1, 1, 2; 3, 5, 8; 13, 21, 33 ]")     # optional - octave
'a =\n\n 1 1 2\n 3 5 8\n 13 21 33\n\n'
sage: octave.eval("b = [ 1; 3; 13]")                         # optional - octave
'b =\n\n 1\n 3\n 13\n\n'
sage: octave.eval("c=a \\ b") # solves linear equation: a*c = b  # optional - octave; random out
'c =\n\n 1\n 7.21645e-16\n -7.21645e-16\n\n'
sage: octave.eval("c")                                 # optional - octave; random output
'c =\n\n 1\n 7.21645e-16\n -7.21645e-16\n\n'
```

`clear`(*var*)

Clear the variable named var.

EXAMPLES:

```
sage: octave.set('x', '2') # optional - octave
sage: octave.clear('x') # optional - octave
sage: octave.get('x') # optional - octave
"error: `x' undefined near line ... column 1"
```

`console`()

Spawn a new Octave command-line session.

This requires that the optional octave program be installed and in your PATH, but no optional Sage packages need be installed.

EXAMPLES:

```
sage: octave_console()         # not tested
GNU Octave, version 2.1.73 (i386-apple-darwin8.5.3).
Copyright (C) 2006 John W. Eaton.
...
octave:1> 2+3
ans = 5
octave:2> [ctl-d]
```

Pressing ctrl-d exits the octave console and returns you to Sage. octave, like Sage, remembers its history from one session to another.

de_system_plot (*f*, *ics*, *trange*)

Plots (using octave's interface to gnuplot) the solution to a 2×2 system of differential equations.

INPUT:

- `f` - a pair of strings representing the differential equations; The independent variable must be called x and the dependent variable must be called y.
- `ics` - a pair [x0,y0] such that x(t0) = x0, y(t0) = y0
- `trange` - a pair [t0,t1]

OUTPUT: a gnuplot window appears

EXAMPLES:

```
sage: octave.de_system_plot(['x+y','x-y'], [1,-1], [0,2])  # not tested -- does this actuall
```

This should yield the two plots $(t, x(t)), (t, y(t))$ on the same graph (the t-axis is the horizontal axis) of the system of ODEs

$$x' = x + y, x(0) = 1; \qquad y' = x - y, y(0) = -1, \quad \text{for} \quad 0 < t < 2.$$

get (*var*)

Get the value of the variable var.

EXAMPLES:

```
sage: octave.set('x', '2') # optional - octave
sage: octave.get('x') # optional - octave
' 2'
```

quit (*verbose=False*)

EXAMPLES:

```
sage: o = Octave()
sage: o._start()    # optional - octave
sage: o.quit(True)  # optional - octave
Exiting spawned Octave process.
```

sage2octave_matrix_string (*A*)

Return an octave matrix from a Sage matrix.

INPUT: A Sage matrix with entries in the rationals or reals.

OUTPUT: A string that evaluates to an Octave matrix.

EXAMPLES:

```
sage: M33 = MatrixSpace(QQ,3,3)
sage: A = M33([1,2,3,4,5,6,7,8,0])
```

```
sage: octave.sage2octave_matrix_string(A)    # optional - octave
'[1, 2, 3; 4, 5, 6; 7, 8, 0]'
```

AUTHORS:

- David Joyner and William Stein

set (*var*, *value*)
Set the variable var to the given value.

EXAMPLES:

```
sage: octave.set('x', '2') # optional - octave
sage: octave.get('x') # optional - octave
' 2'
```

solve_linear_system (*A*, *b*)
Use octave to compute a solution x to A*x = b, as a list.

INPUT:

- `A` - mxn matrix A with entries in QQ or RR
- `b` - m-vector b entries in QQ or RR (resp)

OUTPUT: An list x (if it exists) which solves M*x = b

EXAMPLES:

```
sage: M33 = MatrixSpace(QQ,3,3)
sage: A   = M33([1,2,3,4,5,6,7,8,0])
sage: V3  = VectorSpace(QQ,3)
sage: b   = V3([1,2,3])
sage: octave.solve_linear_system(A,b)    # optional - octave (and output is slightly random
[-0.333332999999999999, 0.666667000000000001, -3.5236600000000002e-18]
```

AUTHORS:

- David Joyner and William Stein

version ()
Return the version of Octave.

OUTPUT: string

EXAMPLES:

```
sage: octave.version()    # optional - octave; random output depending on version
'2.1.73'
```

class sage.interfaces.octave.**OctaveElement** (*parent*, *value*, *is_name=False*, *name=None*)
Bases: `sage.interfaces.expect.ExpectElement`

sage.interfaces.octave.**octave_console** ()
Spawn a new Octave command-line session.

This requires that the optional octave program be installed and in your PATH, but no optional Sage packages need be installed.

EXAMPLES:

```
sage: octave_console()         # not tested
GNU Octave, version 2.1.73 (i386-apple-darwin8.5.3).
Copyright (C) 2006 John W. Eaton.
...
```

```
octave:1> 2+3
ans = 5
octave:2> [ctl-d]
```

Pressing ctrl-d exits the octave console and returns you to Sage. octave, like Sage, remembers its history from one session to another.

`sage.interfaces.octave.octave_version()`

Return the version of Octave installed.

EXAMPLES:

```
sage: octave_version()    # optional - octave; and output is random
'2.9.12'
```

`sage.interfaces.octave.reduce_load_Octave()`

EXAMPLES:

```
sage: from sage.interfaces.octave import reduce_load_Octave
sage: reduce_load_Octave()
Octave
```

CHAPTER

EIGHTEEN

INTERFACE TO R

The following examples try to follow “An Introduction to R” which can be found at http://cran.r-project.org/doc/manuals/R-intro.html .

EXAMPLES:

Simple manipulations; numbers and vectors

The simplest data structure in R is the numeric vector which consists of an ordered collection of numbers. To create a vector named x using the R interface in Sage, you pass the R interpreter object a list or tuple of numbers:

```
sage: x = r([10.4,5.6,3.1,6.4,21.7]); x
[1] 10.4  5.6  3.1  6.4 21.7
```

You can invert elements of a vector x in R by using the invert operator or by doing 1/x:

```
sage: ~x
[1] 0.09615385 0.17857143 0.32258065 0.15625000 0.04608295
sage: 1/x
[1] 0.09615385 0.17857143 0.32258065 0.15625000 0.04608295
```

The following assignment creates a vector y with 11 entries which consists of two copies of x with a 0 in between:

```
sage: y = r([x,0,x]); y
[1] 10.4  5.6  3.1  6.4 21.7  0.0 10.4  5.6  3.1  6.4 21.7
```

Vector Arithmetic

The following command generates a new vector v of length 11 constructed by adding together (element by element) $2x$ repeated 2.2 times, y repeated just once, and 1 repeated 11 times:

```
sage: v = 2*x+y+1; v
[1] 32.2 17.8 10.3 20.2 66.1 21.8 22.6 12.8 16.9 50.8 43.5
```

One can compute the sum of the elements of an R vector in the following two ways:

```
sage: sum(x)
[1] 47.2
sage: x.sum()
[1] 47.2
```

One can calculate the sample variance of a list of numbers:

```
sage: ((x-x.mean())^2/(x.length()-1)).sum()
[1] 53.853
sage: x.var()
```

```
[1] 53.853

sage: x.sort()
[1] 3.1  5.6  6.4 10.4 21.7
sage: x.min()
[1] 3.1
sage: x.max()
[1] 21.7
sage: x
[1] 10.4  5.6  3.1  6.4 21.7

sage: r(-17).sqrt()
[1] NaN
sage: r('-17+0i').sqrt()
[1] 0+4.123106i
```

Generating an arithmetic sequence:

```
sage: r('1:10')
[1]  1  2  3  4  5  6  7  8  9 10
```

Because `from` is a keyword in Python, it can't be used as a keyword argument. Instead, `from_` can be passed, and R will recognize it as the correct thing:

```
sage: r.seq(length=10, from_=-1, by=.2)
[1] -1.0 -0.8 -0.6 -0.4 -0.2  0.0  0.2  0.4  0.6  0.8

sage: x = r([10.4,5.6,3.1,6.4,21.7]);
sage: x.rep(2)
[1] 10.4  5.6  3.1  6.4 21.7 10.4  5.6  3.1  6.4 21.7
sage: x.rep(times=2)
[1] 10.4  5.6  3.1  6.4 21.7 10.4  5.6  3.1  6.4 21.7
sage: x.rep(each=2)
[1] 10.4 10.4  5.6  5.6  3.1  3.1  6.4  6.4 21.7 21.7
```

Missing Values:

```
sage: na = r('NA')
sage: z = r([1,2,3,na])
sage: z
[1]  1  2  3 NA
sage: ind = r.is_na(z)
sage: ind
[1] FALSE FALSE FALSE  TRUE
sage: zero = r(0)
sage: zero / zero
[1] NaN
sage: inf = r('Inf')
sage: inf-inf
[1] NaN
sage: r.is_na(inf)
[1] FALSE
sage: r.is_na(inf-inf)
[1] TRUE
sage: r.is_na(zero/zero)
[1] TRUE
sage: r.is_na(na)
[1] TRUE
```

```
sage: r.is_nan(inf-inf)
[1] TRUE
sage: r.is_nan(zero/zero)
[1] TRUE
sage: r.is_nan(na)
[1] FALSE
```

Character Vectors:

```
sage: labs = r.paste('c("X","Y")', '1:10', sep='""'); labs
[1] "X1"  "Y2"  "X3"  "Y4"  "X5"  "Y6"  "X7"  "Y8"  "X9"  "Y10"
```

Index vectors; selecting and modifying subsets of a data set:

```
sage: na = r('NA')
sage: x = r([10.4,5.6,3.1,6.4,21.7,na]); x
[1] 10.4  5.6  3.1  6.4 21.7   NA
sage: x['!is.na(self)']
[1] 10.4  5.6  3.1  6.4 21.7

sage: x = r([10.4,5.6,3.1,6.4,21.7,na]); x
[1] 10.4  5.6  3.1  6.4 21.7   NA
sage: (x+1)['(!is.na(self)) & self>0']
[1] 11.4  6.6  4.1  7.4 22.7
sage: x = r([10.4,-2,3.1,-0.5,21.7,na]); x
[1] 10.4 -2.0  3.1 -0.5 21.7   NA
sage: (x+1)['(!is.na(self)) & self>0']
[1] 11.4  4.1  0.5 22.7
```

Distributions:

```
sage: r.options(width="60");
$width
[1] 100

sage: rr = r.dnorm(r.seq(-3,3,0.1))
sage: rr
 [1] 0.004431848 0.005952532 0.007915452 0.010420935
 [5] 0.013582969 0.017528300 0.022394530 0.028327038
 [9] 0.035474593 0.043983596 0.053990967 0.065615815
[13] 0.078950158 0.094049077 0.110920835 0.129517596
[17] 0.149727466 0.171368592 0.194186055 0.217852177
[21] 0.241970725 0.266085250 0.289691553 0.312253933
[25] 0.333224603 0.352065327 0.368270140 0.381387815
[29] 0.391042694 0.396952547 0.398942280 0.396952547
[33] 0.391042694 0.381387815 0.368270140 0.352065327
[37] 0.333224603 0.312253933 0.289691553 0.266085250
[41] 0.241970725 0.217852177 0.194186055 0.171368592
[45] 0.149727466 0.129517596 0.110920835 0.094049077
[49] 0.078950158 0.065615815 0.053990967 0.043983596
[53] 0.035474593 0.028327038 0.022394530 0.017528300
[57] 0.013582969 0.010420935 0.007915452 0.005952532
[61] 0.004431848
```

Convert R Data Structures to Python/Sage:

```
sage: rr = r.dnorm(r.seq(-3,3,0.1))
sage: sum(rr._sage_())
```

```
9.9772125168981...
```

Or you get a dictionary to be able to access all the information:

```
sage: rs = r.summary(r.c(1,4,3,4,3,2,5,1))
sage: rs
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.750   3.000   2.875   4.000   5.000
  sage: d = rs._sage_()
  sage: d['DATA']
  [1, 1.75, 3, 2.875, 4, 5]
  sage: d['_Names']
  ['Min.', '1st Qu.', 'Median', 'Mean', '3rd Qu.', 'Max.']
  sage: d['_r_class']
  ['summaryDefault', 'table']
```

It is also possible to access the plotting capabilities of R through Sage. For more information see the documentation of r.plot() or r.png().

AUTHORS:

- Mike Hansen (2007-11-01)
- William Stein (2008-04-19)
- Harald Schilly (2008-03-20)
- Mike Hansen (2008-04-19)

class `sage.interfaces.r.`**`HelpExpression`**

Bases: `str`

Used to improve printing of output of r.help.

class `sage.interfaces.r.`**`R`**(*maxread=100000*, *script_subdirectory=None*, *server_tmpdir=None*, *logfile=None*, *server=None*, *init_list_length=1024*)

Bases: `sage.interfaces.expect.Expect`

An interface to the R interpreter.

R is a comprehensive collection of methods for statistics, modelling, bioinformatics, data analysis and much more. For more details, see http://www.r-project.org/about.html

Resources:

- http://r-project.org/ provides more information about R.
- http://rseek.org/ R's own search engine.

EXAMPLES:

```
sage: r.summary(r.c(1,2,3,111,2,3,2,3,2,5,4))
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
1.00    2.00    3.00   12.55    3.50  111.00
```

TESTS:

```
sage: r == loads(dumps(r))
True
```

`available_packages`()

Returns a list of all available R package names.

This list is not necessarily sorted.

OUTPUT: list of strings

Note: This requires an internet connection. The CRAN server is that is checked is defined at the top of sage/interfaces/r.py.

EXAMPLES:

```
sage: ap = r.available_packages()    # optional - internet
sage: len(ap) > 20                   #optional
True
```

call (*function_name*, **args*, ***kwds*)

This is an alias for `function_call()`.

EXAMPLES:

```
sage: r.call('length', [1,2,3])
[1] 3
```

chdir (*dir*)

Changes the working directory to `dir`

INPUT:

- `dir` – the directory to change to.

EXAMPLES:

```
sage: import tempfile
sage: tmpdir = tempfile.mkdtemp()
sage: r.chdir(tmpdir)
```

Check that `tmpdir` and `r.getwd()` refer to the same directory. We need to use `realpath()` in case `$TMPDIR` (by default `/tmp`) is a symbolic link (see trac ticket #10264).

```
sage: os.path.realpath(tmpdir) == sageobj(r.getwd())   # known bug (:trac:'9970')
True
```

completions (*s*)

Return all commands names that complete the command starting with the string s. This is like typing s[Ctrl-T] in the R interpreter.

INPUT:

- s – string

OUTPUT: list – a list of strings

EXAMPLES:

```
sage: dummy = r.trait_names(use_disk_cache=False)     #clean doctest
sage: r.completions('tes')
['testInheritedMethods', 'testPlatformEquivalence', 'testVirtual']
```

console ()

Runs the R console as a separate new R process.

EXAMPLES:

```
sage: r.console()                    # not tested
    R version 2.6.1 (2007-11-26)
    Copyright (C) 2007 The R Foundation for Statistical Computing
```

```
    ISBN 3-900051-07-0
    ...
```

convert_r_list (*l*)

Converts an R list to a Python list.

EXAMPLES:

```
sage: s = 'c(".GlobalEnv", "package:stats", "package:graphics", "package:grDevices", \n"pack
sage: r.convert_r_list(s)
['.GlobalEnv',
 'package:stats',
 'package:graphics',
 'package:grDevices',
 'package:utils',
 'package:datasets',
 'package:methods',
 'Autoloads',
 'package:base']
```

eval (*code*, *globals=None*, *locals=None*, *synchronize=True*, **args*, ***kwds*)

Evaluates a command inside the R interpreter and returns the output as a string.

EXAMPLES:

```
sage: r.eval('1+1')
'[1] 2'
```

function_call (*function*, *args=None*, *kwds=None*)

Return the result of calling an R function, with given args and keyword args.

OUTPUT: RElement – an object in R

EXAMPLES:

```
sage: r.function_call('length', args=[ [1,2,3] ])
[1] 3
```

get (*var*)

Returns the string representation of the variable var.

INPUT:

- var – a string

OUTPUT: string

EXAMPLES:

```
sage: r.set('a', 2)
sage: r.get('a')
'[1] 2'
```

help (*command*)

Returns help string for a given command.

INPUT: - command – a string

OUTPUT: HelpExpression – a subclass of string whose __repr__ method is __str__, so it prints nicely

EXAMPLES:

```
sage: r.help('c')
c                     package:base                     R Documentation
...

.. note::

This is similar to typing r.command?.
```

install_packages (*package_name*)

Install an R package into Sage's R installation.

EXAMPLES:

```
sage: r.install_packages('aaMI')       # not tested
...
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
...
Please restart Sage in order to use 'aaMI'.
```

library (*library_name*)

Load the library library_name into the R interpreter.

This function raises an ImportError if the given library is not known.

INPUT:

- library_name – string

EXAMPLES:

```
sage: r.library('grid')
sage: 'grid' in r.eval('(.packages())')
True
sage: r.library('foobar')
Traceback (most recent call last):
...
ImportError: ...
```

na ()

Returns the NA in R.

OUTPUT: RElement – an element of R

EXAMPLES:

```
sage: r.na()
[1] NA
```

plot (**args*, ***kwds*)

The R plot function. Type r.help('plot') for much more extensive documentation about this function. See also below for a brief introduction to more plotting with R.

If one simply wants to view an R graphic, using this function is is sufficient (because it calls dev.off() to turn off the device).

However, if one wants to save the graphic to a specific file, it should be used as in the example below to write the output.

EXAMPLES:

This example saves a plot to the standard R output, usually a filename like `Rplot001.png` - from the command line, in the current directory, and in the cell directory in the notebook:

```
sage: d=r.setwd('"%s"'%SAGE_TMP)    # for doctesting only; ignore if you are trying this;
sage: r.plot("1:10")                # optional -- rgraphics
null device
          1
```

To save to a specific file name, one should use `png()` to set the output device to that file. If this is done in the notebook, it must be done in the same cell as the plot itself:

```
sage: filename = tmp_filename() + '.png'
sage: r.png(filename='"%s"'%filename) # Note the double quotes in single quotes!; optional -
NULL
sage: x = r([1,2,3])
sage: y = r([4,5,6])
sage: r.plot(x,y)         # optional -- rgraphics
null device
          1
sage: import os; os.unlink(filename) # For doctesting, we remove the file; optional -- rgrap
```

Please note that for more extensive use of R's plotting capabilities (such as the lattices package), it is advisable to either use an interactive plotting device or to use the notebook. The following examples are not tested, because they differ depending on operating system:

```
sage: r.X11() # not tested - opens interactive device on systems with X11 support
sage: r.quartz() # not tested - opens interactive device on OSX
sage: r.hist("rnorm(100)") # not tested - makes a plot
sage: r.library("lattice") # not tested - loads R lattice plotting package
sage: r.histogram(x = "~ wt | cyl", data="mtcars") # not tested - makes a lattice plot
sage: r.dev_off() # not tested, turns off the interactive viewer
```

In the notebook, one can use r.png() to open the device, but would need to use the following since R lattice graphics do not automatically print away from the command line:

```
sage: filename = tmp_filename() + '.png' # Not needed in notebook, used for doctesting
sage: r.png(filename='"%s"'%filename) # filename not needed in notebook, used for doctesting
NULL
sage: r.library("lattice")
sage: r("print(histogram(~wt | cyl, data=mtcars))") # plot should appear; optional -- rgraph
sage: import os; os.unlink(filename) # We remove the file for doctesting, not needed in note
```

png(**args*, ***kwds*)

Creates an R PNG device.

This should primarily be used to save an R graphic to a custom file. Note that when using this in the notebook, one must plot in the same cell that one creates the device. See r.plot() documentation for more information about plotting via R in Sage.

These examples won't work on the many platforms where R still gets built without graphics support.

EXAMPLES:

```
sage: filename = tmp_filename() + '.png'
sage: r.png(filename='"%s"'%filename)            # optional -- rgraphics
NULL
sage: x = r([1,2,3])
sage: y = r([4,5,6])
sage: r.plot(x,y) # This saves to filename, but is not viewable from command line; optional
null device
          1
```

```
sage: import os; os.unlink(filename) # We remove the file for doctesting; optional -- rgraph
```

We want to make sure that we actually can view R graphics, which happens differently on different platforms:

```
sage: s = r.eval('capabilities("png")') # Should be on Linux and Solaris
sage: t = r.eval('capabilities("aqua")') # Should be on all supported Mac versions
sage: "TRUE" in s+t                     # optional -- rgraphics
True
```

read (*filename*)

Read filename into the R interpreter by calling R's source function on a read-only file connection.

EXAMPLES:

```
sage: filename = tmp_filename()
sage: f = open(filename, 'w')
sage: f.write('a <- 2+2\n')
sage: f.close()
sage: r.read(filename)
sage: r.get('a')
'[1] 4'
```

require (*library_name*)

Load the library library_name into the R interpreter.

This function raises an ImportError if the given library is not known.

INPUT:

- library_name – string

EXAMPLES:

```
sage: r.library('grid')
sage: 'grid' in r.eval('(.packages())')
True
sage: r.library('foobar')
Traceback (most recent call last):
...
ImportError: ...
```

set (*var*, *value*)

Set the variable var in R to what the string value evaluates to in R.

INPUT:

- var – a string
- value – a string

EXAMPLES:

```
sage: r.set('a', '2 + 3')
sage: r.get('a')
'[1] 5'
```

source (*s*)

Display the R source (if possible) about the function named s.

INPUT:

- s – a string representing the function whose source code you want to see

OUTPUT: string – source code

EXAMPLES:

```
sage: print r.source("c")
function (..., recursive = FALSE)  .Primitive("c")
```

trait_names (*verbose=True*, *use_disk_cache=True*)
Return list of all R functions.

INPUT:

- verbose – bool (default: True); if True, display debugging information
- use_disk_cache – bool (default: True); if True, use the disk cache of trait names to save time.

OUTPUT: list – list of string

EXAMPLES:

```
sage: t = r.trait_names(verbose=False)
sage: len(t) > 200
True
```

version ()
Return the version of R currently running.

OUTPUT: tuple of ints; string

EXAMPLES:

```
sage: r.version() # not tested
((3, 0, 1), 'R version 3.0.1 (2013-05-16)')
sage: rint, rstr = r.version()
sage: rint[0] >= 3
True
sage: rstr.startswith('R version')
True
```

class sage.interfaces.r.**RElement** (*parent*, *value*, *is_name=False*, *name=None*)
Bases: sage.interfaces.expect.ExpectElement

dot_product (*other*)
Implements the notation self . other.

INPUT:

- self, other – R elements

OUTPUT: R element

EXAMPLES:

```
sage: c = r.c(1,2,3,4)
sage: c.dot_product(c.t())
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    2    4    6    8
[3,]    3    6    9   12
[4,]    4    8   12   16

sage: v = r([3,-1,8])
sage: v.dot_product(v)
     [,1]
[1,]   74
```

stat_model (*x*)
The tilde regression operator in R.

EXAMPLES:

```
sage: x = r([1,2,3,4,5])
sage: y = r([3,5,7,9,11])
sage: a = r.lm( y.tilde(x) ) # lm( y ~ x )
sage: d = a._sage_()
sage: d['DATA']['coefficients']['DATA'][1]
2
```

tilde (*x*)
The tilde regression operator in R.

EXAMPLES:

```
sage: x = r([1,2,3,4,5])
sage: y = r([3,5,7,9,11])
sage: a = r.lm( y.tilde(x) ) # lm( y ~ x )
sage: d = a._sage_()
sage: d['DATA']['coefficients']['DATA'][1]
2
```

trait_names ()
Return a list of all methods of this object.

Note: Currently returns all R commands.

EXAMPLES:

```
sage: a = r([1,2,3])
sage: t = a.trait_names()
sage: len(t) > 200
True
```

class sage.interfaces.r.**RFunction** (*parent*, *name*, *r_name=None*)
Bases: sage.interfaces.expect.ExpectFunction

A Function in the R interface.

INPUT:

- parent – the R interface
- name – the name of the function for Python
- r_name – the name of the function in R itself (which can have dots in it)

EXAMPLES:

```
sage: length = r.length
sage: type(length)
<class 'sage.interfaces.r.RFunction'>
sage: loads(dumps(length))
length
```

class sage.interfaces.r.**RFunctionElement** (*obj*, *name*)
Bases: sage.interfaces.expect.FunctionElement

sage.interfaces.r.**is_RElement** (*x*)
Return True if x is an element in an R interface.

INPUT:

- x – object

OUTPUT: bool

EXAMPLES:

```
sage: from sage.interfaces.r import is_RElement
sage: is_RElement(2)
False
sage: is_RElement(r(2))
True
```

`sage.interfaces.r.`**`r_console`**`()`

Spawn a new R command-line session.

EXAMPLES:

```
sage: r.console()                    # not tested
    R version 2.6.1 (2007-11-26)
    Copyright (C) 2007 The R Foundation for Statistical Computing
    ISBN 3-900051-07-0
    ...
```

`sage.interfaces.r.`**`r_version`**`()`

Return the R version.

EXAMPLES:

```
sage: r_version() # not tested
((3, 0, 1), 'R version 3.0.1 (2013-05-16)')
sage: rint, rstr = r_version()
sage: rint[0] >= 3
True
sage: rstr.startswith('R version')
True
```

`sage.interfaces.r.`**`reduce_load_R`**`()`

Used for reconstructing a copy of the R interpreter from a pickle.

EXAMPLES:

```
sage: from sage.interfaces.r import reduce_load_R
sage: reduce_load_R()
R Interpreter
```

CHAPTER

NINETEEN

INTERFACE TO SAGE

This is an expect interface to *another* copy of the Sage interpreter.

class `sage.interfaces.sage0.`**`Sage`**(*logfile=None*, *preparse=True*, *python=False*, *init_code=None*, *server=None*, *server_tmpdir=None*, *remote_cleaner=True*, ***kwds*)

Bases: `sage.interfaces.expect.Expect`

Expect interface to the Sage interpreter itself.

INPUT:

- `server` - (optional); if specified runs Sage on a remote machine with address. You must have ssh keys setup so you can login to the remote machine by typing "ssh remote_machine" and no password, call _install_hints_ssh() for hints on how to do that.

The version of Sage should be the same as on the local machine, since pickling is used to move data between the two Sage process.

EXAMPLES: We create an interface to a copy of Sage. This copy of Sage runs as an external process with its own memory space, etc.

```
sage: s = Sage()
```

Create the element 2 in our new copy of Sage, and cube it.

```
sage: a = s(2)
sage: a^3
8
```

Create a vector space of dimension 4, and compute its generators:

```
sage: V = s('QQ^4')
sage: V.gens()
((1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1))
```

Note that V is a not a vector space, it's a wrapper around an object (which happens to be a vector space), in another running instance of Sage.

```
sage: type(V)
<class 'sage.interfaces.sage0.SageElement'>
sage: V.parent()
Sage
sage: g = V.0;  g
(1, 0, 0, 0)
sage: g.parent()
Sage
```

We can still get the actual parent by using the name attribute of g, which is the variable name of the object in the child process.

```
sage: s('%s.parent()'%g.name())
Vector space of dimension 4 over Rational Field
```

Note that the memory space is completely different.

```
sage: x = 10
sage: s('x = 5')
5
sage: x
10
sage: s('x')
5
```

We can have the child interpreter itself make another child Sage process, so now three copies of Sage are running:

```
sage: s3 = s('Sage()')
sage: a = s3(10)
sage: a
10
```

This $a = 10$ is in a subprocess of a subprocesses of your original Sage.

```
sage: _ = s.eval('%s.eval("x=8")'%s3.name())
sage: s3('"x"')
8
sage: s('x')
5
sage: x
10
```

The double quotes are needed because the call to s3 first evaluates its arguments using the s interpreter, so the call to s3 is passed `s('"x"')`, which is the string `"x"` in the s interpreter.

clear(*var*)

Clear the variable named var.

Note that the exact format of the NameError for a cleared variable is slightly platform dependent, see trac #10539.

EXAMPLES:

```
sage: sage0.set('x', '2')
sage: sage0.get('x')
'2'
sage: sage0.clear('x')
sage: 'NameError' in sage0.get('x')
True
```

console()

Spawn a new Sage command-line session.

EXAMPLES:

```
sage: sage0.console() #not tested
----------------------------------------------------------------------
| Sage Version ..., Release Date: ...                                |
| Type notebook() for the GUI, and license() for information.        |
----------------------------------------------------------------------
```

```
...
```

cputime (*t=None*)

Return cputime since this Sage subprocess was started.

EXAMPLES:

```
sage: sage0.cputime()     # random output
1.3530439999999999
sage: sage0('factor(2^157-1)')
852133201 * 60726444167 * 1654058017289 * 2134387368610417
sage: sage0.cputime()     # random output
1.6462939999999999
```

eval (*line*, *strip=True*, ***kwds*)

Send the code x to a second instance of the Sage interpreter and return the output as a string.

This allows you to run two completely independent copies of Sage at the same time in a unified way.

INPUT:

- `line` - input line of code
- `strip` - ignored

EXAMPLES:

```
sage: sage0.eval('2+2')
'4'
```

get (*var*)

Get the value of the variable var.

EXAMPLES:

```
sage: sage0.set('x', '2')
sage: sage0.get('x')
'2'
```

new (*x*)

EXAMPLES:

```
sage: sage0.new(2)
2
sage: _.parent()
Sage
```

preparse (*x*)

Returns the preparsed version of the string s.

EXAMPLES:

```
sage: sage0.preparse('2+2')
'Integer(2)+Integer(2)'
```

quit (*verbose=False*)

EXAMPLES:

```
sage: s = Sage()
sage: s.eval('2+2')
'4'
sage: s.quit()
```

set (*var*, *value*)

Set the variable var to the given value.

EXAMPLES:

```
sage: sage0.set('x', '2')
sage: sage0.get('x')
'2'
```

trait_names ()

EXAMPLES:

```
sage: t = sage0.trait_names()
sage: len(t) > 100
True
sage: 'gcd' in t
True
```

version ()

EXAMPLES:

```
sage: sage0.version()
'Sage Version ..., Release Date: ...'
sage: sage0.version() == version()
True
```

class sage.interfaces.sage0.**SageElement** (*parent*, *value*, *is_name=False*, *name=None*)

Bases: sage.interfaces.expect.ExpectElement

class sage.interfaces.sage0.**SageFunction** (*obj*, *name*)

Bases: sage.interfaces.expect.FunctionElement

sage.interfaces.sage0.**reduce_load_Sage** ()

EXAMPLES:

```
sage: from sage.interfaces.sage0 import reduce_load_Sage
sage: reduce_load_Sage()
Sage
```

sage.interfaces.sage0.**reduce_load_element** (*s*)

EXAMPLES:

```
sage: from sage.interfaces.sage0 import reduce_load_element
sage: s = dumps(1/2)
sage: half = reduce_load_element(s); half
1/2
sage: half.parent()
Sage
```

sage.interfaces.sage0.**sage0_console** ()

Spawn a new Sage command-line session.

EXAMPLES:

```
sage: sage0_console() #not tested
----------------------------------------------------------------------
| Sage Version ..., Release Date: ...                                |
| Type notebook() for the GUI, and license() for information.        |
----------------------------------------------------------------------
...
```

sage.interfaces.sage0.**sage0_version** ()

EXAMPLES:

```
sage: from sage.interfaces.sage0 import sage0_version
sage: sage0_version() == version()
True
```

CHAPTER

TWENTY

INTERFACE TO SINGULAR

AUTHORS:

- David Joyner and William Stein (2005): first version
- Martin Albrecht (2006-03-05): code so singular.[tab] and x = singular(...), x.[tab] includes all singular commands.
- Martin Albrecht (2006-03-06): This patch adds the equality symbol to singular. Also fix a problem in which " " as prompt means comparison will break all further communication with Singular.
- Martin Albrecht (2006-03-13): added current_ring() and current_ring_name()
- William Stein (2006-04-10): Fixed problems with ideal constructor
- Martin Albrecht (2006-05-18): added sage_poly.
- Simon King (2010-11-23): Reduce the overhead caused by waiting for the Singular prompt by doing garbage collection differently.
- Simon King (2011-06-06): Make conversion from Singular to Sage more flexible.

20.1 Introduction

This interface is extremely flexible, since it's exactly like typing into the Singular interpreter, and anything that works there should work here.

The Singular interface will only work if Singular is installed on your computer; this should be the case, since Singular is included with Sage. The interface offers three pieces of functionality:

1. `singular_console()` - A function that dumps you into an interactive command-line Singular session.
2. `singular(expr, type='def')` - Creation of a Singular object. This provides a Pythonic interface to Singular. For example, if `f=singular(10)`, then `f.factorize()` returns the factorization of 10 computed using Singular.
3. `singular.eval(expr)` - Evaluation of arbitrary Singular expressions, with the result returned as a string.

Of course, there are polynomial rings and ideals in Sage as well (often based on a C-library interface to Singular). One can convert an object in the Singular interpreter interface to Sage by the method `sage()`.

20.2 Tutorial

EXAMPLES: First we illustrate multivariate polynomial factorization:

```
sage: R1 = singular.ring(0, '(x,y)', 'dp')
sage: R1
//   characteristic : 0
//   number of vars : 2
//        block   1 : ordering dp
//                  : names    x y
//        block   2 : ordering C
sage: f = singular('9x16 - 18x13y2 - 9x12y3 + 9x10y4 - 18x11y2 + 36x8y4 + 18x7y5 - 18x5y6 + 9x6y4 - 1
sage: f
9*x^16-18*x^13*y^2-9*x^12*y^3+9*x^10*y^4-18*x^11*y^2+36*x^8*y^4+18*x^7*y^5-18*x^5*y^6+9*x^6*y^4-18*x
sage: f.parent()
Singular
```

```
sage: F = f.factorize(); F
[1]:
   _[1]=9
   _[2]=x^6-2*x^3*y^2-x^2*y^3+y^4
   _[3]=-x^5+y^2
[2]:
   1,1,2
```

```
sage: F[1]
9,
x^6-2*x^3*y^2-x^2*y^3+y^4,
-x^5+y^2
sage: F[1][2]
x^6-2*x^3*y^2-x^2*y^3+y^4
```

We can convert f and each exponent back to Sage objects as well.

```
sage: g = f.sage(); g
9*x^16 - 18*x^13*y^2 - 9*x^12*y^3 + 9*x^10*y^4 - 18*x^11*y^2 + 36*x^8*y^4 + 18*x^7*y^5 - 18*x^5*y^6 +
sage: F[1][2].sage()
x^6 - 2*x^3*y^2 - x^2*y^3 + y^4
sage: g.parent()
Multivariate Polynomial Ring in x, y over Rational Field
```

This example illustrates polynomial GCD's:

```
sage: R2 = singular.ring(0, '(x,y,z)', 'lp')
sage: a = singular.new('3x2*(x+y)')
sage: b = singular.new('9x*(y2-x2)')
sage: g = a.gcd(b)
sage: g
x^2+x*y
```

This example illustrates computation of a Groebner basis:

```
sage: R3 = singular.ring(0, '(a,b,c,d)', 'lp')
sage: I = singular.ideal(['a + b + c + d', 'a*b + a*d + b*c + c*d', 'a*b*c + a*b*d + a*c*d + b*c*d',
sage: I2 = I.groebner()
sage: I2
c^2*d^6-c^2*d^2-d^4+1,
c^3*d^2+c^2*d^3-c-d,
b*d^4-b+d^5-d,
b*c-b*d^5+c^2*d^4+c*d-d^6-d^2,
b^2+2*b*d+d^2,
```

```
a+b+c+d
```

The following example is the same as the one in the Singular - Gap interface documentation:

```
sage: R  = singular.ring(0, '(x0,x1,x2)', 'lp')
sage: I1 = singular.ideal(['x0*x1*x2 -x0^2*x2', 'x0^2*x1*x2-x0*x1^2*x2-x0*x1*x2^2', 'x0*x1-x0*x2-x1*
sage: I2 = I1.groebner()
sage: I2
x1^2*x2^2,
x0*x2^3-x1^2*x2^2+x1*x2^3,
x0*x1-x0*x2-x1*x2,
x0^2*x2-x0*x2^2-x1*x2^2
sage: I2.sage()
Ideal (x1^2*x2^2, x0*x2^3 - x1^2*x2^2 + x1*x2^3, x0*x1 - x0*x2 - x1*x2, x0^2*x2 - x0*x2^2 - x1*x2^2)
```

This example illustrates moving a polynomial from one ring to another. It also illustrates calling a method of an object with an argument.

```
sage: R = singular.ring(0, '(x,y,z)', 'dp')
sage: f = singular('x3+y3+(x-y)*x2y2+z2')
sage: f
x^3*y^2-x^2*y^3+x^3+y^3+z^2
sage: R1 = singular.ring(0, '(x,y,z)', 'ds')
sage: f = R.fetch(f)
sage: f
z^2+x^3+y^3+x^3*y^2-x^2*y^3
```

We can calculate the Milnor number of f:

```
sage: _=singular.LIB('sing.lib')      # assign to _ to suppress printing
sage: f.milnor()
4
```

The Jacobian applied twice yields the Hessian matrix of f, with which we can compute.

```
sage: H = f.jacob().jacob()
sage: H
6*x+6*x*y^2-2*y^3,6*x^2*y-6*x*y^2,  0,
6*x^2*y-6*x*y^2,  6*y+2*x^3-6*x^2*y,0,
0,                0,                2
sage: H.sage()
[6*x + 6*x*y^2 - 2*y^3     6*x^2*y - 6*x*y^2                     0]
[    6*x^2*y - 6*x*y^2 6*y + 2*x^3 - 6*x^2*y                     0]
[                    0                     0                     2]
sage: H.det()   # This is a polynomial in Singular
72*x*y+24*x^4-72*x^3*y+72*x*y^3-24*y^4-48*x^4*y^2+64*x^3*y^3-48*x^2*y^4
sage: H.det().sage()   # This is the corresponding polynomial in Sage
72*x*y + 24*x^4 - 72*x^3*y + 72*x*y^3 - 24*y^4 - 48*x^4*y^2 + 64*x^3*y^3 - 48*x^2*y^4
```

The 1x1 and 2x2 minors:

```
sage: H.minor(1)
2,
6*y+2*x^3-6*x^2*y,
6*x^2*y-6*x*y^2,
6*x^2*y-6*x*y^2,
6*x+6*x*y^2-2*y^3
sage: H.minor(2)
```

```
12*y+4*x^3-12*x^2*y,
12*x^2*y-12*x*y^2,
12*x^2*y-12*x*y^2,
12*x+12*x*y^2-4*y^3,
-36*x*y-12*x^4+36*x^3*y-36*x*y^3+12*y^4+24*x^4*y^2-32*x^3*y^3+24*x^2*y^4
```

```
sage: _=singular.eval('option(redSB)')
sage: H.minor(1).groebner()
1
```

20.3 Computing the Genus

We compute the projective genus of ideals that define curves over **Q**. It is *very important* to load the `normal.lib` library before calling the `genus` command, or you'll get an error message.

EXAMPLE:

```
sage: singular.lib('normal.lib')
sage: R = singular.ring(0,'(x,y)','dp')
sage: i2 = singular.ideal('y9 - x2*(x-1)^9 + x')
sage: i2.genus()
40
```

Note that the genus can be much smaller than the degree:

```
sage: i = singular.ideal('y9 - x2*(x-1)^9')
sage: i.genus()
0
```

20.4 An Important Concept

AUTHORS:

- Neal Harris

The following illustrates an important concept: how Sage interacts with the data being used and returned by Singular. Let's compute a Groebner basis for some ideal, using Singular through Sage.

```
sage: singular.lib('poly.lib')
sage: singular.ring(32003, '(a,b,c,d,e,f)', 'lp')
        //   characteristic : 32003
        //   number of vars : 6
        //        block   1 : ordering lp
        //                          : names    a b c d e f
        //        block   2 : ordering C
sage: I = singular.ideal('cyclic(6)')
sage: g = singular('groebner(I)')
Traceback (most recent call last):
...
TypeError: Singular error:
...
```

We restart everything and try again, but correctly.

```
sage: singular.quit()
sage: singular.lib('poly.lib'); R = singular.ring(32003, '(a,b,c,d,e,f)', 'lp')
sage: I = singular.ideal('cyclic(6)')
sage: I.groebner()
f^48-2554*f^42-15674*f^36+12326*f^30-12326*f^18+15674*f^12+2554*f^6-1,
...
```

It's important to understand why the first attempt at computing a basis failed. The line where we gave singular the input 'groebner(I)' was useless because Singular has no idea what 'I' is! Although 'I' is an object that we computed with calls to Singular functions, it actually lives in Sage. As a consequence, the name 'I' means nothing to Singular. When we called `I.groebner()`, Sage was able to call the groebner function on'I' in Singular, since 'I' actually means something to Sage.

20.5 Long Input

The Singular interface reads in even very long input (using files) in a robust manner, as long as you are creating a new object.

```
sage: t = '"%s"'%10^15000   # 15 thousand character string (note that normal Singular input must be a
sage: a = singular.eval(t)
sage: a = singular(t)
```

TESTS:

We test an automatic coercion:

```
sage: a = 3*singular('2'); a
6
sage: type(a)
<class 'sage.interfaces.singular.SingularElement'>
sage: a = singular('2')*3; a
6
sage: type(a)
<class 'sage.interfaces.singular.SingularElement'>
```

Create a ring over GF(9) to check that `gftables` has been installed, see ticket #11645:

```
sage: singular.eval("ring testgf9 = (9,x),(a,b,c,d,e,f),(M((1,2,3,0)),wp(2,3),lp);")
'ring testgf9 = (9,x),(a,b,c,d,e,f),(M((1,2,3,0)),wp(2,3),lp);'
```

class `sage.interfaces.singular.`**`Singular`**(*maxread=1000*, *script_subdirectory=None*, *logfile=None*, *server=None*, *server_tmpdir=None*)

Bases: `sage.interfaces.expect.Expect`

Interface to the Singular interpreter.

EXAMPLES: A Groebner basis example.

```
sage: R = singular.ring(0, '(x0,x1,x2)', 'lp')
sage: I = singular.ideal([ 'x0*x1*x2 -x0^2*x2', 'x0^2*x1*x2-x0*x1^2*x2-x0*x1*x2^2', 'x0*x1-x0*x2
sage: I.groebner()
x1^2*x2^2,
x0*x2^3-x1^2*x2^2+x1*x2^3,
x0*x1-x0*x2-x1*x2,
x0^2*x2-x0*x2^2-x1*x2^2
```

AUTHORS:

•David Joyner and William Stein

LIB (*lib*, *reload=False*)

Load the Singular library named lib.

Note that if the library was already loaded during this session it is not reloaded unless the optional reload argument is True (the default is False).

EXAMPLES:

```
sage: singular.lib('sing.lib')
sage: singular.lib('sing.lib', reload=True)
```

clear (*var*)

Clear the variable named `var`.

EXAMPLES:

```
sage: singular.set('int', 'x', '2')
sage: singular.get('x')
'2'
sage: singular.clear('x')
```

"Clearing the variable" means to allow to free the memory that it uses in the Singular sub-process. However, the actual deletion of the variable is only committed when the next element in the Singular interface is created:

```
sage: singular.get('x')
'2'
sage: a = singular(3)
sage: singular.get('x')
'`x`'
```

console ()

EXAMPLES:

```
sage: singular_console() #not tested
                     SINGULAR                             /  Development
 A Computer Algebra System for Polynomial Computations   /   version 3-0-4
                                                       0<
     by: G.-M. Greuel, G. Pfister, H. Schoenemann        \   Nov 2007
FB Mathematik der Universitaet, D-67653 Kaiserslautern
```

cputime (*t=None*)

Returns the amount of CPU time that the Singular session has used. If `t` is not None, then it returns the difference between the current CPU time and `t`.

EXAMPLES:

```
sage: t = singular.cputime()
sage: R = singular.ring(0, '(x0,x1,x2)', 'lp')
sage: I = singular.ideal([ 'x0*x1*x2 -x0^2*x2', 'x0^2*x1*x2-x0*x1^2*x2-x0*x1*x2^2', 'x0*x1-x
sage: gb = I.groebner()
sage: singular.cputime(t) #random
0.02
```

current_ring ()

Returns the current ring of the running Singular session.

EXAMPLES:

```
sage: r = PolynomialRing(GF(127),3,'xyz', order='invlex')
sage: r._singular_()
//   characteristic : 127
//   number of vars : 3
//        block   1 : ordering rp
//                  : names    x y z
//        block   2 : ordering C
sage: singular.current_ring()
//   characteristic : 127
//   number of vars : 3
//        block   1 : ordering rp
//                  : names    x y z
//        block   2 : ordering C
```

current_ring_name()

Returns the Singular name of the currently active ring in Singular.

OUTPUT: currently active ring's name

EXAMPLES:

```
sage: r = PolynomialRing(GF(127),3,'xyz')
sage: r._singular_().name() == singular.current_ring_name()
True
```

eval(*x*, *allow_semicolon=True*, *strip=True*, ***kwds*)

Send the code x to the Singular interpreter and return the output as a string.

INPUT:

- `x` - string (of code)
- `allow_semicolon` - default: False; if False then raise a TypeError if the input line contains a semicolon.
- `strip` - ignored

EXAMPLES:

```
sage: singular.eval('2 > 1')
'1'
sage: singular.eval('2 + 2')
'4'
```

if the verbosity level is > 1 comments are also printed and not only returned.

```
sage: r = singular.ring(0,'(x,y,z)','dp')
sage: i = singular.ideal(['x^2','y^2','z^2'])
sage: s = i.std()
sage: singular.eval('hilb(%s)'%(s.name()))
'// 1 t^0\n// -3 t^2\n// 3 t^4\n// -1 t^6\n\n// 1 t^0\n//
3 t^1\n// 3 t^2\n// 1 t^3\n// dimension (affine) = 0\n//
degree (affine) = 8'

sage: set_verbose(1)
sage: o = singular.eval('hilb(%s)'%(s.name()))
//         1 t^0
//        -3 t^2
//         3 t^4
//        -1 t^6
//         1 t^0
//         3 t^1
```

```
//          3 t^2
//          1 t^3
// dimension (affine) = 0
// degree (affine)  = 8
```

This is mainly useful if this method is called implicitly. Because then intermediate results, debugging outputs and printed statements are printed

```
sage: o = s.hilb()
//          1 t^0
//         -3 t^2
//          3 t^4
//         -1 t^6
//          1 t^0
//          3 t^1
//          3 t^2
//          1 t^3
// dimension (affine) = 0
// degree (affine)  = 8
// ** right side is not a datum, assignment ignored
```

rather than ignored

```
sage: set_verbose(0)
sage: o = s.hilb()
```

get (*var*)

Get string representation of variable named var.

EXAMPLES:

```
sage: singular.set('int', 'x', '2')
sage: singular.get('x')
'2'
```

has_coerce_map_from_impl (*S*)

x.__init__(...) initializes x; see help(type(x)) for signature

ideal (**gens*)

Return the ideal generated by gens.

INPUT:

- gens - list or tuple of Singular objects (or objects that can be made into Singular objects via evaluation)

OUTPUT: the Singular ideal generated by the given list of gens

EXAMPLES: A Groebner basis example done in a different way.

```
sage: _ = singular.eval("ring R=0,(x0,x1,x2),lp")
sage: i1 = singular.ideal([ 'x0*x1*x2 -x0^2*x2', 'x0^2*x1*x2-x0*x1^2*x2-x0*x1*x2^2', 'x0*x1-
sage: i1
-x0^2*x2+x0*x1*x2,
x0^2*x1*x2-x0*x1^2*x2-x0*x1*x2^2,
x0*x1-x0*x2-x1*x2

sage: i2 = singular.ideal('groebner(%s);'%i1.name())
sage: i2
x1^2*x2^2,
x0*x2^3-x1^2*x2^2+x1*x2^3,
```

```
x0*x1-x0*x2-x1*x2,
x0^2*x2-x0*x2^2-x1*x2^2
```

lib (*lib*, *reload=False*)

Load the Singular library named lib.

Note that if the library was already loaded during this session it is not reloaded unless the optional reload argument is True (the default is False).

EXAMPLES:

```
sage: singular.lib('sing.lib')
sage: singular.lib('sing.lib', reload=True)
```

list (*x*)

Creates a list in Singular from a Sage list x.

EXAMPLES:

```
sage: singular.list([1,2])
[1]:
   1
[2]:
   2
```

load (*lib*, *reload=False*)

Load the Singular library named lib.

Note that if the library was already loaded during this session it is not reloaded unless the optional reload argument is True (the default is False).

EXAMPLES:

```
sage: singular.lib('sing.lib')
sage: singular.lib('sing.lib', reload=True)
```

matrix (*nrows*, *ncols*, *entries=None*)

EXAMPLES:

```
sage: singular.lib("matrix")
sage: R = singular.ring(0, '(x,y,z)', 'dp')
sage: A = singular.matrix(3,2,'1,2,3,4,5,6')
sage: A
1,2,
3,4,
5,6
sage: A.gauss_col()
2,-1,
1,0,
0,1
```

AUTHORS:

- Martin Albrecht (2006-01-14)

option (*cmd=None*, *val=None*)

Access to Singular's options as follows:

Syntax: option() Returns a string of all defined options.

Syntax: option('option_name') Sets an option. Note to disable an option, use the prefix no.

Syntax: option('get') Returns an intvec of the state of all options.

Syntax: option('set', intvec_expression) Restores the state of all options from an intvec (produced by option('get')).

EXAMPLES:

```
sage: singular.option()
//options: redefine loadLib usage prompt
sage: singular.option('get')
0,
10321
sage: old_options = _
sage: singular.option('noredefine')
sage: singular.option()
//options: loadLib usage prompt
sage: singular.option('set', old_options)
sage: singular.option('get')
0,
10321
```

ring (*char=0*, *vars='(x)'*, *order='lp'*, *check=True*)

Create a Singular ring and makes it the current ring.

INPUT:

- `char` - characteristic of the base ring (see examples below), which must be either 0, prime (!), or one of several special codes (see examples below).
- `vars` - a tuple or string that defines the variable names
- `order` - string - the monomial order (default: 'lp')
- `check` - if True, check primality of the characteristic if it is an integer.

OUTPUT: a Singular ring

Note: This function is *not* identical to calling the Singular `ring` function. In particular, it also attempts to "kill" the variable names, so they can actually be used without getting errors, and it sets printing of elements for this range to short (i.e., with *'s and carets).

EXAMPLES: We first declare $\mathbf{Q}[x, y, z]$ with degree reverse lexicographic ordering.

```
sage: R = singular.ring(0, '(x,y,z)', 'dp')
sage: R
//   characteristic : 0
//   number of vars : 3
//        block   1 : ordering dp
//                  : names    x y z
//        block   2 : ordering C

sage: R1 = singular.ring(32003, '(x,y,z)', 'dp')
sage: R2 = singular.ring(32003, '(a,b,c,d)', 'lp')
```

This is a ring in variables named x(1) through x(10) over the finite field of order 7:

```
sage: R3 = singular.ring(7, '(x(1..10))', 'ds')
```

This is a polynomial ring over the transcendental extension $\mathbf{Q}(a)$ of $\mathbf{Q}$:

```
sage: R4 = singular.ring('(0,a)', '(mu,nu)', 'lp')
```

This is a ring over the field of single-precision floats:

```
sage: R5 = singular.ring('real', '(a,b)', 'lp')
```

This is over 50-digit floats:

```
sage: R6 = singular.ring('(real,50)', '(a,b)', 'lp')
sage: R7 = singular.ring('(complex,50,i)', '(a,b)', 'lp')
```

To use a ring that you've defined, use the set_ring() method on the ring. This sets the ring to be the "current ring". For example,

```
sage: R = singular.ring(7, '(a,b)', 'ds')
sage: S = singular.ring('real', '(a,b)', 'lp')
sage: singular.new('10*a')
1.000e+01*a
sage: R.set_ring()
sage: singular.new('10*a')
3*a
```

set (*type*, *name*, *value*)

Set the variable with given name to the given value.

REMARK:

If a variable in the Singular interface was previously marked for deletion, the actual deletion is done here, before the new variable is created in Singular.

EXAMPLES:

```
sage: singular.set('int', 'x', '2')
sage: singular.get('x')
'2'
```

We test that an unused variable is only actually deleted if this method is called:

```
sage: a = singular(3)
sage: n = a.name()
sage: del a
sage: singular.eval(n)
'3'
sage: singular.set('int', 'y', '5')
sage: singular.eval('defined(%s)'%n)
'0'
```

set_ring (*R*)

Sets the current Singular ring to R.

EXAMPLES:

```
sage: R = singular.ring(7, '(a,b)', 'ds')
sage: S = singular.ring('real', '(a,b)', 'lp')
sage: singular.current_ring()
//   characteristic : 0 (real)
//   number of vars : 2
//        block   1 : ordering lp
//                  : names    a b
//        block   2 : ordering C
sage: singular.set_ring(R)
sage: singular.current_ring()
//   characteristic : 7
//   number of vars : 2
```

```
//        block   1 : ordering ds
//                  : names    a b
//        block   2 : ordering C
```

setring(*R*)

Sets the current Singular ring to R.

EXAMPLES:

```
sage: R = singular.ring(7, '(a,b)', 'ds')
sage: S = singular.ring('real', '(a,b)', 'lp')
sage: singular.current_ring()
//   characteristic : 0 (real)
//   number of vars : 2
//        block   1 : ordering lp
//                  : names    a b
//        block   2 : ordering C
sage: singular.set_ring(R)
sage: singular.current_ring()
//   characteristic : 7
//   number of vars : 2
//        block   1 : ordering ds
//                  : names    a b
//        block   2 : ordering C
```

string(*x*)

Creates a Singular string from a Sage string. Note that the Sage string has to be "double-quoted".

EXAMPLES:

```
sage: singular.string('"Sage"')
Sage
```

trait_names()

Return a list of all Singular commands.

EXAMPLES:

```
sage: singular.trait_names()
['exteriorPower',
 ...
 'stdfglm']
```

version()

EXAMPLES:

class sage.interfaces.singular.**SingularElement**(*parent*, *type*, *value*, *is_name=False*)

Bases: sage.interfaces.expect.ExpectElement

EXAMPLES:

```
sage: a = singular(2)
sage: loads(dumps(a))
(invalid object -- defined in terms of closed session)
```

attrib(*name*, *value=None*)

Get and set attributes for self.

INPUT:

- name - string to choose the attribute

- `value` - boolean value or None for reading, (default:None)

VALUES: isSB - the standard basis property is set by all commands computing a standard basis like groebner, std, stdhilb etc.; used by lift, dim, degree, mult, hilb, vdim, kbase isHomog - the weight vector for homogeneous or quasihomogeneous ideals/modules isCI - complete intersection property isCM - Cohen-Macaulay property rank - set the rank of a module (see nrows) withSB - value of type ideal, resp. module, is std withHilb - value of type intvec is hilb(_,1) (see hilb) withRes - value of type list is a free resolution withDim - value of type int is the dimension (see dim) withMult - value of type int is the multiplicity (see mult)

EXAMPLE:

```
sage: P.<x,y,z> = PolynomialRing(QQ)
sage: I = Ideal([z^2, y*z, y^2, x*z, x*y, x^2])
sage: Ibar = I._singular_()
sage: Ibar.attrib('isSB')
0
sage: singular.eval('vdim(%s)'%Ibar.name()) # sage7 name is random
// ** sage7 is no standard basis
4
sage: Ibar.attrib('isSB',1)
sage: singular.eval('vdim(%s)'%Ibar.name())
'4'
```

sage_flattened_str_list()

EXAMPLES:

```
sage: R=singular.ring(0,'(x,y)','dp')
sage: RL = R.ringlist()
sage: RL.sage_flattened_str_list()
['0', 'x', 'y', 'dp', '1,1', 'C', '0', '_[1]=0']
```

sage_global_ring()

Return the current basering in Singular as a polynomial ring or quotient ring.

EXAMPLE:

```
sage: singular.eval('ring r1 = (9,x),(a,b,c,d,e,f),(M((1,2,3,0)),wp(2,3),lp)')
'ring r1 = (9,x),(a,b,c,d,e,f),(M((1,2,3,0)),wp(2,3),lp);'
sage: R = singular('r1').sage_global_ring()
sage: R
Multivariate Polynomial Ring in a, b, c, d, e, f over Finite Field in x of size 3^2
sage: R.term_order()
Block term order with blocks:
(Matrix term order with matrix
[1 2]
[3 0],
 Weighted degree reverse lexicographic term order with weights (2, 3),
 Lexicographic term order of length 2)

sage: singular.eval('ring r2 = (0,x),(a,b,c),dp')
'ring r2 = (0,x),(a,b,c),dp;'
sage: singular('r2').sage_global_ring()
Multivariate Polynomial Ring in a, b, c over Fraction Field of Univariate Polynomial Ring in

sage: singular.eval('ring r3 = (3,z),(a,b,c),dp')
'ring r3 = (3,z),(a,b,c),dp;'
sage: singular.eval('minpoly = 1+z+z2+z3+z4')
'minpoly = 1+z+z2+z3+z4;'
sage: singular('r3').sage_global_ring()
Multivariate Polynomial Ring in a, b, c over Finite Field in z of size 3^4
```

Real and complex fields in both Singular and Sage are defined with a precision. The precision in Singular is given in terms of digits, but in Sage it is given in terms of bits. So, the digit precision is internally converted to a reasonable bit precision:

```
sage: singular.eval('ring r4 = (real,20),(a,b,c),dp')
'ring r4 = (real,20),(a,b,c),dp;'
sage: singular('r4').sage_global_ring()
Multivariate Polynomial Ring in a, b, c over Real Field with 70 bits of precision
```

The case of complex coefficients is not fully supported, yet, since the generator of a complex field in Sage is always called "I":

```
sage: singular.eval('ring r5 = (complex,15,j),(a,b,c),dp')
'ring r5 = (complex,15,j),(a,b,c),dp;'
sage: R = singular('r5').sage_global_ring(); R
Multivariate Polynomial Ring in a, b, c over Complex Field with 54 bits of precision
sage: R.base_ring()('j')
Traceback (most recent call last):
...
NameError: name 'j' is not defined
sage: R.base_ring()('I')
1.00000000000000*I
```

In our last example, the base ring is a quotient ring:

```
sage: singular.eval('ring r6 = (9,a), (x,y,z),lp')
'ring r6 = (9,a), (x,y,z),lp;'
sage: Q = singular('std(ideal(x^2,x+y^2+z^3))', type='qring')
sage: Q.sage_global_ring()
Quotient of Multivariate Polynomial Ring in x, y, z over Finite Field in a of size 3^2 by th
```

AUTHOR:

- Simon King (2011-06-06)

sage_matrix (*R*, *sparse=True*)

Returns Sage matrix for self

INPUT:

- R - (default: None); an optional ring, over which the resulting matrix is going to be defined. By default, the output of `sage_global_ring()` is used.
- sparse - (default: True); determines whether the resulting matrix is sparse or not.

EXAMPLES:

```
sage: R = singular.ring(0, '(x,y,z)', 'dp')
sage: A = singular.matrix(2,2)
sage: A.sage_matrix(ZZ)
[0 0]
[0 0]
sage: A.sage_matrix(RDF)
[0.0 0.0]
[0.0 0.0]
```

sage_poly (*R=None*, *kcache=None*)

Returns a Sage polynomial in the ring r matching the provided poly which is a singular polynomial.

INPUT:

- R - (default: None); an optional polynomial ring. If it is provided, then you have to make sure that it matches the current singular ring as, e.g., returned by singular.current_ring(). By default, the output of `sage_global_ring()` is used.

- `kcache` - (default: None); an optional dictionary for faster finite field lookups, this is mainly useful for finite extension fields

OUTPUT: MPolynomial

EXAMPLES:

```
sage: R = PolynomialRing(GF(2^8,'a'),2,'xy')
sage: f=R('a^20*x^2*y+a^10+x')
sage: f._singular_().sage_poly(R)==f
True
sage: R = PolynomialRing(GF(2^8,'a'),1,'x')
sage: f=R('a^20*x^3+x^2+a^10')
sage: f._singular_().sage_poly(R)==f
True

sage: P.<x,y> = PolynomialRing(QQ, 2)
sage: f = x*y**3 - 1/9 * x + 1; f
x*y^3 - 1/9*x + 1
sage: singular(f)
x*y^3-1/9*x+1
sage: P(singular(f))
x*y^3 - 1/9*x + 1
```

TESTS:

```
sage: singular.eval('ring r = (3,z),(a,b,c),dp')
'ring r = (3,z),(a,b,c),dp;'
sage: singular.eval('minpoly = 1+z+z2+z3+z4')
'minpoly = 1+z+z2+z3+z4;'
sage: p = singular('z^4*a^3+z^2*a*b*c')
sage: p.sage_poly()
(-z^3 - z^2 - z - 1)*a^3 + (z^2)*a*b*c
sage: singular('z^4')
(-z3-z2-z-1)
```

AUTHORS:

- Martin Albrecht (2006-05-18)
- Simon King (2011-06-06): Deal with Singular's short polynomial representation, automatic construction of a polynomial ring, if it is not explicitly given.

Note: For very simple polynomials `eval(SingularElement.sage_polystring())` is faster than SingularElement.sage_poly(R), maybe we should detect the crossover point (in dependence of the string length) and choose an appropriate conversion strategy

---

sage_polystring()

If this Singular element is a polynomial, return a string representation of this polynomial that is suitable for evaluation in Python. Thus * is used for multiplication and ** for exponentiation. This function is primarily used internally.

The short=0 option *must* be set for the parent ring or this function will not work as expected. This option is set by default for rings created using `singular.ring` or set using `ring_name.set_ring()`.

EXAMPLES:

```
sage: R = singular.ring(0,'(x,y)')
sage: f = singular('x^3 + 3*y^11 + 5')
sage: f
x^3+3*y^11+5
sage: f.sage_polystring()
'x**3+3*y**11+5'
```

sage_structured_str_list()

If self is a Singular list of lists of Singular elements, returns corresponding Sage list of lists of strings.

EXAMPLES:

```
sage: R=singular.ring(0,'(x,y)','dp')
sage: RL=R.ringlist()
sage: RL
[1]:
   0
[2]:
   [1]:
      x
   [2]:
      y
[3]:
   [1]:
      [1]:
         dp
      [2]:
         1,1
   [2]:
      [1]:
         C
      [2]:
         0
[4]:
   _[1]=0
sage: RL.sage_structured_str_list()
['0', ['x', 'y'], [['dp', '1,\n1 '], ['C', '0 ']], '0']
```

set_ring()

Sets the current ring in Singular to be self.

EXAMPLES:

```
sage: R = singular.ring(7, '(a,b)', 'ds')
sage: S = singular.ring('real', '(a,b)', 'lp')
sage: singular.current_ring()
//   characteristic : 0 (real)
//   number of vars : 2
//        block   1 : ordering lp
//                  : names    a b
//        block   2 : ordering C
sage: R.set_ring()
sage: singular.current_ring()
//   characteristic : 7
//   number of vars : 2
//        block   1 : ordering ds
//                  : names    a b
//        block   2 : ordering C
```

trait_names()

Returns the possible tab-completions for self. In this case, we just return all the tab completions for the Singular object.

EXAMPLES:

```
sage: R = singular.ring(0,'(x,y)','dp')
sage: R.trait_names()
['exteriorPower',
 ...
 'stdfglm']
```

type()

Returns the internal type of this element.

EXAMPLES:

```
sage: R = PolynomialRing(GF(2^8,'a'),2,'x')
sage: R._singular_().type()
'ring'
sage: fs = singular('x0^2','poly')
sage: fs.type()
'poly'
```

exception sage.interfaces.singular.**SingularError**

Bases: exceptions.RuntimeError

Raised if Singular printed an error message

class sage.interfaces.singular.**SingularFunction**(*parent*, *name*)

Bases: sage.interfaces.expect.ExpectFunction

class sage.interfaces.singular.**SingularFunctionElement**(*obj*, *name*)

Bases: sage.interfaces.expect.FunctionElement

class sage.interfaces.singular.**SingularGBLogPrettyPrinter**(*verbosity=1*)

A device which prints Singular Groebner basis computation logs more verbatim.

flush()

EXAMPLE:

```
sage: from sage.interfaces.singular import SingularGBLogPrettyPrinter
sage: s3 = SingularGBLogPrettyPrinter(verbosity=3)
sage: s3.flush()
```

write(*s*)

EXAMPLE:

```
sage: from sage.interfaces.singular import SingularGBLogPrettyPrinter
sage: s3 = SingularGBLogPrettyPrinter(verbosity=3)
sage: s3.write("(S:1337)")
Performing complete reduction of 1337 elements.
sage: s3.write("M[389,12]")
Parallel reduction of 389 elements with 12 non-zero output elements.
```

sage.interfaces.singular.**generate_docstring_dictionary**()

Generate global dictionaries which hold the docstrings for Singular functions.

EXAMPLE:

```
sage: from sage.interfaces.singular import generate_docstring_dictionary
sage: generate_docstring_dictionary()
```

`sage.interfaces.singular.`**`get_docstring`**`(`*name*`)`

Return the docstring for the function `name`.

INPUT:

- `name` - a Singular function name

EXAMPLE:

```
sage: from sage.interfaces.singular import get_docstring
sage: 'groebner' in get_docstring('groebner')
True
sage: 'standard.lib' in get_docstring('groebner')
True
```

`sage.interfaces.singular.`**`is_SingularElement`**`(`*x*`)`

Returns True is x is of type `SingularElement`.

EXAMPLES:

```
sage: from sage.interfaces.singular import is_SingularElement
sage: is_SingularElement(singular(2))
True
sage: is_SingularElement(2)
False
```

`sage.interfaces.singular.`**`reduce_load`**`()`

Note that this returns an invalid Singular object!

EXAMPLES:

```
sage: from sage.interfaces.singular import reduce_load
sage: reduce_load()
(invalid object -- defined in terms of closed session)
```

`sage.interfaces.singular.`**`reduce_load_Singular`**`()`

EXAMPLES:

```
sage: from sage.interfaces.singular import reduce_load_Singular
sage: reduce_load_Singular()
Singular
```

`sage.interfaces.singular.`**`singular_console`**`()`

Spawn a new Singular command-line session.

EXAMPLES:

```
sage: singular_console() #not tested
                     SINGULAR                             /  Development
 A Computer Algebra System for Polynomial Computations   /   version 3-0-4
                                                       0<
     by: G.-M. Greuel, G. Pfister, H. Schoenemann        \   Nov 2007
FB Mathematik der Universitaet, D-67653 Kaiserslautern
```

`sage.interfaces.singular.`**`singular_version`**`()`

Returns the version of Singular being used.

EXAMPLES:

CHAPTER

TWENTYONE

THE TACHYON RAY TRACER

AUTHOR:

- John E. Stone

class `sage.interfaces.tachyon.`**`TachyonRT`**

The Tachyon Ray Tracer

tachyon_rt(model, outfile='sage.png', verbose=1, block=True, extra_opts='')

INPUT:

- `model` - a string that describes a 3d model in the Tachyon modeling format. Type tachyon_rt.help() for a description of this format.
- `outfile` - (default: 'sage.png') output filename; the extension of the filename determines the type. Supported types include:
  - `tga` - 24-bit (uncompressed)
  - `bmp` - 24-bit Windows BMP (uncompressed)
  - `ppm` - 24-bit PPM (uncompressed)
  - `rgb` - 24-bit SGI RGB (uncompressed)
  - `png` - 24-bit PNG (compressed, lossless)
- `verbose` - integer; (default: 1)
  - `0` - silent
  - `1` - some output
  - `2` - very verbose output
- `block` - bool (default: True); if False, run the rendering command in the background.
- `extra_opts` - passed directly to tachyon command line. Use tachyon_rt.usage() to see some of the possibilities.

OUTPUT:

- Some text may be displayed onscreen.
- The file outfile is created.

EXAMPLES:

AUTHORS:

- John E. Stone

help(*use_pager=True*)

Prints (pages) the help file written by John Stone describing scene files for Tachyon. The output is paged unless use_pager=False.

TESTS:

```
sage: from sage.interfaces.tachyon import TachyonRT
sage: t = TachyonRT()
sage: t.help(use_pager=False)
This help, which was written by John Stone, describes ...
```

usage(*use_pager=True*)

Returns the basic description of using the Tachyon raytracer (simply what is returned by running tachyon with no input). The output is paged unless use_pager=False.

TESTS:

```
sage: from sage.interfaces.tachyon import TachyonRT
sage: t = TachyonRT()
sage: t.usage(use_pager=False)
Tachyon Parallel/Multiprocessor Ray Tracer   Version...
```

CHAPTER

TWENTYTWO

INTERFACE FOR EXTRACTING DATA AND GENERATING IMAGES FROM JMOL READABLE FILES.

JmolData is a no GUI version of Jmol useful for extracting data from files Jmol reads and for generating image files.

AUTHORS:

- Jonathan Gutow (2012-06-14): complete doctest coverage
- Jonathan Gutow (2012-03-21): initial version

class `sage.interfaces.jmoldata.`**`JmolData`**

Bases: `sage.structure.sage_object.SageObject`

Todo

Create an animated image file (GIF) if spin is on and put data extracted from a file into a variable/string/structure to return

`export_image`(*targetfile*, *datafile*, *datafile_cmd='script'*, *image_type='PNG'*, *figsize=5*, ***kwds*)

This executes JmolData.jar to make an image file.

INPUT:

- targetfile – the full path to the file where the image should be written.
- datafile – full path to the data file Jmol can read or text of a script telling Jmol what to read or load.
- datafile_cmd – (default `'script'`) `'load'` or `'script'` should be `"load"` for a data file.
- image_type – (default `"PNG"`) `'PNG'` `'JPG'` or `'GIF'`
- figsize – number (default 5) equal to (pixels/side)/100

OUTPUT:

Image file, .png, .gif or .jpg (default .png)

Note: Examples will generate an error message if a functional Java Virtual Machine (JVM) is not installed on the machine the Sage instance is running on.

Warning: Programmers using this module should check that the JVM is available before making calls to avoid the user getting error messages. Check for the JVM using the function `is_jvm_available()`, which returns True if a JVM is available.

EXAMPLES:

Use Jmol to load a pdb file containing some DNA from a web data base and make an image of the DNA. If you execute this in the notebook, the image will appear in the output cell:

```
sage: from sage.interfaces.jmoldata import JmolData
sage: JData = JmolData()
sage: script = "load =1lcd;display DNA;moveto 0.0 { -473 -713 -518 59.94} 100.0 0.0 0.0 {21.
sage: testfile = tmp_filename(ext="DNA.png")
sage: JData.export_image(targetfile=testfile,datafile=script,image_type="PNG")  # optional -
sage: print os.path.exists(testfile)  # optional -- java internet
True
```

Use Jmol to save an image of a 3-D object created in Sage. This method is used internally by plot3d to generate static images. This example doesn't have correct scaling:

```
sage: from sage.interfaces.jmoldata import JmolData
sage: JData = JmolData()
sage: D=dodecahedron()
sage: from sage.misc.misc import SAGE_TMP
sage: archive_name=os.path.join(SAGE_TMP, "archive.jmol.zip")
sage: D.export_jmol(archive_name)  #not scaled properly...need some more steps.
sage: testfile = os.path.join(SAGE_TMP, "testimage.png")
sage: script = 'set defaultdirectory "%s"\n script SCRIPT\n'%archive_name
sage: JData.export_image(targetfile =testfile,datafile = script, image_type="PNG") # optiona
sage: print os.path.exists(testfile) # optional -- java
True
```

is_jvm_available()

Returns True if the Java Virtual Machine is available and False if not.

EXAMPLES:

Check that it returns a boolean:

```
sage: from sage.interfaces.jmoldata import JmolData
sage: JData = JmolData()
sage: type(JData.is_jvm_available())
<type 'bool'>
```

CHAPTER

TWENTYTHREE

INDICES AND TABLES

- Index
- Module Index
- Search Page

PYTHON MODULE INDEX

i

sage.interfaces.axiom, 7
sage.interfaces.ecm, 13
sage.interfaces.expect, 3
sage.interfaces.four_ti_2, 19
sage.interfaces.gap, 25
sage.interfaces.gap3, 37
sage.interfaces.gnuplot, 57
sage.interfaces.gp, 47
sage.interfaces.jmoldata, 177
sage.interfaces.kash, 59
sage.interfaces.magma, 67
sage.interfaces.maple, 85
sage.interfaces.mathematica, 121
sage.interfaces.matlab, 93
sage.interfaces.maxima, 97
sage.interfaces.maxima_lib, 107
sage.interfaces.mwrank, 129
sage.interfaces.octave, 133
sage.interfaces.r, 139
sage.interfaces.sage0, 151
sage.interfaces.singular, 157
sage.interfaces.tachyon, 175

INDEX

A

add_vararg() (in module sage.interfaces.maxima_lib), 115
as_type() (sage.interfaces.axiom.PanAxiomElement method), 11
assign_names() (sage.interfaces.magma.MagmaElement method), 78
AssignNames() (sage.interfaces.magma.MagmaElement method), 78
Attach() (sage.interfaces.magma.Magma method), 70
attach() (sage.interfaces.magma.Magma method), 71
attach_spec() (sage.interfaces.magma.Magma method), 72
AttachSpec() (sage.interfaces.magma.Magma method), 71
attrib() (sage.interfaces.singular.SingularElement method), 168
available_packages() (sage.interfaces.r.R method), 142
Axiom (class in sage.interfaces.axiom), 9
axiom_console() (in module sage.interfaces.axiom), 12
AxiomElement (class in sage.interfaces.axiom), 9
AxiomExpectFunction (class in sage.interfaces.axiom), 9
AxiomFunctionElement (class in sage.interfaces.axiom), 9

B

bar_call() (sage.interfaces.magma.Magma method), 72
bool() (sage.interfaces.gap.GapElement_generic method), 29
bool() (sage.interfaces.gp.GpElement method), 54

C

call() (sage.interfaces.four_ti_2.FourTi2 method), 19
call() (sage.interfaces.r.R method), 143
chdir() (sage.interfaces.magma.Magma method), 73
chdir() (sage.interfaces.mathematica.Mathematica method), 126
chdir() (sage.interfaces.matlab.Matlab method), 95
chdir() (sage.interfaces.r.R method), 143
circuits() (sage.interfaces.four_ti_2.FourTi2 method), 19
clean_output() (in module sage.interfaces.mathematica), 127
clear() (sage.interfaces.magma.Magma method), 73
clear() (sage.interfaces.maple.Maple method), 88
clear() (sage.interfaces.maxima.Maxima method), 103
clear() (sage.interfaces.maxima_lib.MaximaLib method), 108
clear() (sage.interfaces.octave.Octave method), 135

clear() (sage.interfaces.sage0.Sage method), 152
clear() (sage.interfaces.singular.Singular method), 162
clear_prompts() (sage.interfaces.expect.Expect method), 3
comma() (sage.interfaces.axiom.PanAxiomElement method), 11
completions() (sage.interfaces.maple.Maple method), 88
completions() (sage.interfaces.r.R method), 143
console() (in module sage.interfaces.expect), 5
console() (sage.interfaces.axiom.Axiom method), 9
console() (sage.interfaces.gap.Gap method), 27
console() (sage.interfaces.gap3.Gap3 method), 43
console() (sage.interfaces.gnuplot.Gnuplot method), 57
console() (sage.interfaces.gp.Gp method), 49
console() (sage.interfaces.kash.Kash method), 65
console() (sage.interfaces.magma.Magma method), 73
console() (sage.interfaces.maple.Maple method), 88
console() (sage.interfaces.mathematica.Mathematica method), 126
console() (sage.interfaces.matlab.Matlab method), 95
console() (sage.interfaces.mwrank.Mwrank_class method), 129
console() (sage.interfaces.octave.Octave method), 135
console() (sage.interfaces.r.R method), 143
console() (sage.interfaces.sage0.Sage method), 152
console() (sage.interfaces.singular.Singular method), 162
convert_r_list() (sage.interfaces.r.R method), 144
cputime() (sage.interfaces.gap.Gap method), 27
cputime() (sage.interfaces.gap3.Gap3 method), 43
cputime() (sage.interfaces.gp.Gp method), 49
cputime() (sage.interfaces.magma.Magma method), 74
cputime() (sage.interfaces.maple.Maple method), 88
cputime() (sage.interfaces.sage0.Sage method), 153
cputime() (sage.interfaces.singular.Singular method), 162
current_ring() (sage.interfaces.singular.Singular method), 162
current_ring_name() (sage.interfaces.singular.Singular method), 163

D

de_system_plot() (sage.interfaces.octave.Octave method), 136
directory() (sage.interfaces.four_ti_2.FourTi2 method), 19
display2d() (sage.interfaces.maxima.MaximaElement method), 104
display2d() (sage.interfaces.maxima_lib.MaximaLibElement method), 114
dot_product() (sage.interfaces.r.RElement method), 148
dummy_integrate() (in module sage.interfaces.maxima_lib), 115

E

ecl() (sage.interfaces.maxima_lib.MaximaLibElement method), 114
ECM (class in sage.interfaces.ecm), 13
eval() (sage.interfaces.expect.Expect method), 3
eval() (sage.interfaces.gap.Gap_generic method), 29
eval() (sage.interfaces.kash.Kash method), 65
eval() (sage.interfaces.magma.Magma method), 74
eval() (sage.interfaces.magma.MagmaElement method), 79
eval() (sage.interfaces.mathematica.Mathematica method), 126

eval() (sage.interfaces.maxima_lib.MaximaLib method), 108
eval() (sage.interfaces.mwrank.Mwrank_class method), 129
eval() (sage.interfaces.r.R method), 144
eval() (sage.interfaces.sage0.Sage method), 153
eval() (sage.interfaces.singular.Singular method), 163
evaluate() (sage.interfaces.magma.MagmaElement method), 79
Expect (class in sage.interfaces.expect), 3
expect() (sage.interfaces.expect.Expect method), 4
expect() (sage.interfaces.maple.Maple method), 88
ExpectElement (class in sage.interfaces.expect), 5
ExpectFunction (class in sage.interfaces.expect), 5
export_image() (sage.interfaces.jmoldata.JmolData method), 177
extcode_dir() (in module sage.interfaces.magma), 83

F

factor() (sage.interfaces.ecm.ECM method), 14
find_factor() (sage.interfaces.ecm.ECM method), 15
flush() (sage.interfaces.magma.MagmaGBLogPrettyPrinter method), 83
flush() (sage.interfaces.singular.SingularGBLogPrettyPrinter method), 173
FourTi2 (class in sage.interfaces.four_ti_2), 19
function_call() (sage.interfaces.gap.Gap_generic method), 30
function_call() (sage.interfaces.magma.Magma method), 75
function_call() (sage.interfaces.r.R method), 144
FunctionElement (class in sage.interfaces.expect), 5

G

Gap (class in sage.interfaces.gap), 27
Gap3 (class in sage.interfaces.gap3), 42
gap3_console() (in module sage.interfaces.gap3), 44
gap3_version() (in module sage.interfaces.gap3), 44
GAP3Element (class in sage.interfaces.gap3), 40
GAP3Record (class in sage.interfaces.gap3), 41
gap_command() (in module sage.interfaces.gap), 32
gap_console() (in module sage.interfaces.gap), 32
Gap_generic (class in sage.interfaces.gap), 29
gap_reset_workspace() (in module sage.interfaces.gap), 32
GapElement (class in sage.interfaces.gap), 28
GapElement_generic (class in sage.interfaces.gap), 29
GapFunction (class in sage.interfaces.gap), 29
GapFunctionElement (class in sage.interfaces.gap), 29
gc_disabled (class in sage.interfaces.expect), 5
gen() (sage.interfaces.magma.MagmaElement method), 79
gen_names() (sage.interfaces.magma.MagmaElement method), 80
generate_docstring_dictionary() (in module sage.interfaces.singular), 173
gens() (sage.interfaces.magma.MagmaElement method), 80
get() (sage.interfaces.axiom.PanAxiom method), 10
get() (sage.interfaces.gap.Gap method), 28
get() (sage.interfaces.gp.Gp method), 50
get() (sage.interfaces.kash.Kash method), 65
get() (sage.interfaces.magma.Magma method), 75

get() (sage.interfaces.maple.Maple method), 89
get() (sage.interfaces.mathematica.Mathematica method), 126
get() (sage.interfaces.matlab.Matlab method), 95
get() (sage.interfaces.maxima.Maxima method), 103
get() (sage.interfaces.maxima_lib.MaximaLib method), 109
get() (sage.interfaces.octave.Octave method), 136
get() (sage.interfaces.r.R method), 144
get() (sage.interfaces.sage0.Sage method), 153
get() (sage.interfaces.singular.Singular method), 164
get_default() (sage.interfaces.gp.Gp method), 50
get_docstring() (in module sage.interfaces.singular), 173
get_gap_memory_pool_size() (in module sage.interfaces.gap), 33
get_last_params() (sage.interfaces.ecm.ECM method), 16
get_magma_attribute() (sage.interfaces.magma.MagmaElement method), 80
get_precision() (sage.interfaces.gp.Gp method), 50
get_real_precision() (sage.interfaces.gp.Gp method), 50
get_record_element() (sage.interfaces.gap.Gap_generic method), 30
get_series_precision() (sage.interfaces.gp.Gp method), 50
get_verbose() (sage.interfaces.magma.Magma method), 75
GetVerbose() (sage.interfaces.magma.Magma method), 71
gfq_gap_to_sage() (in module sage.interfaces.gap), 33
Gnuplot (class in sage.interfaces.gnuplot), 57
gnuplot() (sage.interfaces.gnuplot.Gnuplot method), 57
gnuplot_console() (in module sage.interfaces.gnuplot), 58
Gp (class in sage.interfaces.gp), 49
gp_console() (in module sage.interfaces.gp), 54
gp_version() (in module sage.interfaces.gp), 54
GpElement (class in sage.interfaces.gp), 53
GpFunction (class in sage.interfaces.gp), 54
GpFunctionElement (class in sage.interfaces.gp), 54
graver() (sage.interfaces.four_ti_2.FourTi2 method), 20
groebner() (sage.interfaces.four_ti_2.FourTi2 method), 20

H

has_coerce_map_from_impl() (sage.interfaces.singular.Singular method), 164
help() (sage.interfaces.gap.Gap method), 28
help() (sage.interfaces.gap3.Gap3 method), 43
help() (sage.interfaces.gp.Gp method), 51
help() (sage.interfaces.kash.Kash method), 65
help() (sage.interfaces.magma.Magma method), 76
help() (sage.interfaces.maple.Maple method), 89
help() (sage.interfaces.mathematica.Mathematica method), 127
help() (sage.interfaces.r.R method), 144
help() (sage.interfaces.tachyon.TachyonRT method), 175
help_search() (sage.interfaces.kash.Kash method), 65
HelpExpression (class in sage.interfaces.r), 142
hilbert() (sage.interfaces.four_ti_2.FourTi2 method), 20

I

ideal() (sage.interfaces.magma.Magma method), 76

ideal() (sage.interfaces.magma.MagmaElement method), 81
ideal() (sage.interfaces.singular.Singular method), 164
install_packages() (sage.interfaces.r.R method), 145
interact() (sage.interfaces.ecm.ECM method), 16
interact() (sage.interfaces.gnuplot.Gnuplot method), 57
interrupt() (sage.interfaces.expect.Expect method), 4
interrupt() (sage.interfaces.gap.Gap_generic method), 31
intmod_gap_to_sage() (in module sage.interfaces.gap), 34
is_AxiomElement() (in module sage.interfaces.axiom), 12
is_ExpectElement() (in module sage.interfaces.expect), 5
is_GapElement() (in module sage.interfaces.gap), 34
is_GpElement() (in module sage.interfaces.gp), 54
is_jvm_available() (sage.interfaces.jmoldata.JmolData method), 178
is_KashElement() (in module sage.interfaces.kash), 66
is_local() (sage.interfaces.expect.Expect method), 4
is_MagmaElement() (in module sage.interfaces.magma), 83
is_MaximaElement() (in module sage.interfaces.maxima), 105
is_MaximaLibElement() (in module sage.interfaces.maxima_lib), 115
is_RElement() (in module sage.interfaces.r), 149
is_remote() (sage.interfaces.expect.Expect method), 4
is_running() (sage.interfaces.expect.Expect method), 4
is_SingularElement() (in module sage.interfaces.singular), 174

J

JmolData (class in sage.interfaces.jmoldata), 177

K

Kash (class in sage.interfaces.kash), 65
kash_console() (in module sage.interfaces.kash), 66
kash_version() (in module sage.interfaces.kash), 66
KashDocumentation (class in sage.interfaces.kash), 66
KashElement (class in sage.interfaces.kash), 66
kill() (sage.interfaces.gp.Gp method), 51

L

LIB() (sage.interfaces.singular.Singular method), 162
lib() (sage.interfaces.singular.Singular method), 165
library() (sage.interfaces.r.R method), 145
lisp() (sage.interfaces.maxima.Maxima method), 104
lisp() (sage.interfaces.maxima_lib.MaximaLib method), 109
list() (sage.interfaces.singular.Singular method), 165
list_attributes() (sage.interfaces.magma.MagmaElement method), 81
load() (sage.interfaces.magma.Magma method), 76
load() (sage.interfaces.maple.Maple method), 89
load() (sage.interfaces.singular.Singular method), 165
load_package() (sage.interfaces.gap.Gap_generic method), 31

M

Magma (class in sage.interfaces.magma), 70
magma_console() (in module sage.interfaces.magma), 84

magma_version() (in module sage.interfaces.magma), 84
MagmaElement (class in sage.interfaces.magma), 78
MagmaFunction (class in sage.interfaces.magma), 83
MagmaFunctionElement (class in sage.interfaces.magma), 83
MagmaGBLogPrettyPrinter (class in sage.interfaces.magma), 83
Maple (class in sage.interfaces.maple), 87
maple_console() (in module sage.interfaces.maple), 90
MapleElement (class in sage.interfaces.maple), 90
MapleFunction (class in sage.interfaces.maple), 90
MapleFunctionElement (class in sage.interfaces.maple), 90
Mathematica (class in sage.interfaces.mathematica), 126
mathematica_console() (in module sage.interfaces.mathematica), 127
MathematicaElement (class in sage.interfaces.mathematica), 127
MathematicaFunction (class in sage.interfaces.mathematica), 127
MathematicaFunctionElement (class in sage.interfaces.mathematica), 127
Matlab (class in sage.interfaces.matlab), 95
matlab_console() (in module sage.interfaces.matlab), 96
matlab_version() (in module sage.interfaces.matlab), 96
MatlabElement (class in sage.interfaces.matlab), 96
matrix() (sage.interfaces.singular.Singular method), 165
max_at_to_sage() (in module sage.interfaces.maxima_lib), 116
max_to_sr() (in module sage.interfaces.maxima_lib), 116
max_to_string() (in module sage.interfaces.maxima_lib), 116
Maxima (class in sage.interfaces.maxima), 103
MaximaElement (class in sage.interfaces.maxima), 104
MaximaElementFunction (class in sage.interfaces.maxima), 104
MaximaFunction (class in sage.interfaces.maxima), 105
MaximaFunctionElement (class in sage.interfaces.maxima), 105
MaximaLib (class in sage.interfaces.maxima_lib), 107
MaximaLibElement (class in sage.interfaces.maxima_lib), 113
MaximaLibElementFunction (class in sage.interfaces.maxima_lib), 114
MaximaLibFunction (class in sage.interfaces.maxima_lib), 115
MaximaLibFunctionElement (class in sage.interfaces.maxima_lib), 115
mdiff_to_sage() (in module sage.interfaces.maxima_lib), 117
methods() (sage.interfaces.magma.MagmaElement method), 81
minimize() (sage.interfaces.four_ti_2.FourTi2 method), 20
mlist_to_sage() (in module sage.interfaces.maxima_lib), 117
mqapply_to_sage() (in module sage.interfaces.maxima_lib), 117
mrat_to_sage() (in module sage.interfaces.maxima_lib), 117
mul_vararg() (in module sage.interfaces.maxima_lib), 118
Mwrank() (in module sage.interfaces.mwrank), 129
Mwrank_class (class in sage.interfaces.mwrank), 129
mwrank_console() (in module sage.interfaces.mwrank), 130

N

N() (sage.interfaces.mathematica.MathematicaElement method), 127
na() (sage.interfaces.r.R method), 145
new() (sage.interfaces.sage0.Sage method), 153
new_with_bits_prec() (sage.interfaces.gp.Gp method), 51

O

objgens() (sage.interfaces.magma.Magma method), 77
Octave (class in sage.interfaces.octave), 135
octave_console() (in module sage.interfaces.octave), 137
octave_version() (in module sage.interfaces.octave), 138
OctaveElement (class in sage.interfaces.octave), 137
one_curve() (sage.interfaces.ecm.ECM method), 16
operations() (sage.interfaces.gap3.GAP3Record method), 41
option() (sage.interfaces.singular.Singular method), 165

P

PanAxiom (class in sage.interfaces.axiom), 10
PanAxiomElement (class in sage.interfaces.axiom), 10
PanAxiomExpectFunction (class in sage.interfaces.axiom), 12
PanAxiomFunctionElement (class in sage.interfaces.axiom), 12
parse_max_string() (in module sage.interfaces.maxima_lib), 118
path() (sage.interfaces.expect.Expect method), 4
pid() (sage.interfaces.expect.Expect method), 4
plot() (sage.interfaces.gnuplot.Gnuplot method), 57
plot() (sage.interfaces.r.R method), 145
plot3d() (sage.interfaces.gnuplot.Gnuplot method), 57
plot3d_parametric() (sage.interfaces.gnuplot.Gnuplot method), 57
png() (sage.interfaces.r.R method), 146
ppi() (sage.interfaces.four_ti_2.FourTi2 method), 21
preparse() (sage.interfaces.sage0.Sage method), 153
pyobject_to_max() (in module sage.interfaces.maxima_lib), 118

Q

qsolve() (sage.interfaces.four_ti_2.FourTi2 method), 21
quit() (sage.interfaces.expect.Expect method), 4
quit() (sage.interfaces.gp.Gp method), 51
quit() (sage.interfaces.mwrank.Mwrank_class method), 130
quit() (sage.interfaces.octave.Octave method), 136
quit() (sage.interfaces.sage0.Sage method), 153
quo() (sage.interfaces.magma.MagmaElement method), 81

R

R (class in sage.interfaces.r), 142
r_console() (in module sage.interfaces.r), 150
r_version() (in module sage.interfaces.r), 150
rays() (sage.interfaces.four_ti_2.FourTi2 method), 21
read() (sage.interfaces.expect.Expect method), 4
read() (sage.interfaces.r.R method), 147
read_matrix() (sage.interfaces.four_ti_2.FourTi2 method), 21
recfields() (sage.interfaces.gap3.GAP3Record method), 41
recommended_B1() (sage.interfaces.ecm.ECM method), 17
reduce_load() (in module sage.interfaces.gap), 34
reduce_load() (in module sage.interfaces.mathematica), 127
reduce_load() (in module sage.interfaces.singular), 174
reduce_load_Axiom() (in module sage.interfaces.axiom), 12

reduce_load_element() (in module sage.interfaces.sage0), 154
reduce_load_GAP() (in module sage.interfaces.gap), 35
reduce_load_GP() (in module sage.interfaces.gp), 55
reduce_load_Kash() (in module sage.interfaces.kash), 66
reduce_load_Magma() (in module sage.interfaces.magma), 84
reduce_load_Maple() (in module sage.interfaces.maple), 91
reduce_load_Matlab() (in module sage.interfaces.matlab), 96
reduce_load_Maxima() (in module sage.interfaces.maxima), 105
reduce_load_Maxima_function() (in module sage.interfaces.maxima), 105
reduce_load_MaximaLib() (in module sage.interfaces.maxima_lib), 119
reduce_load_Octave() (in module sage.interfaces.octave), 138
reduce_load_R() (in module sage.interfaces.r), 150
reduce_load_Sage() (in module sage.interfaces.sage0), 154
reduce_load_Singular() (in module sage.interfaces.singular), 174
RElement (class in sage.interfaces.r), 148
require() (sage.interfaces.r.R method), 147
RFunction (class in sage.interfaces.r), 149
RFunctionElement (class in sage.interfaces.r), 149
ring() (sage.interfaces.singular.Singular method), 166

S

Sage (class in sage.interfaces.sage0), 151
sage.interfaces.axiom (module), 7
sage.interfaces.ecm (module), 13
sage.interfaces.expect (module), 3
sage.interfaces.four_ti_2 (module), 19
sage.interfaces.gap (module), 25
sage.interfaces.gap3 (module), 37
sage.interfaces.gnuplot (module), 57
sage.interfaces.gp (module), 47
sage.interfaces.jmoldata (module), 177
sage.interfaces.kash (module), 59
sage.interfaces.magma (module), 67
sage.interfaces.maple (module), 85
sage.interfaces.mathematica (module), 121
sage.interfaces.matlab (module), 93
sage.interfaces.maxima (module), 97
sage.interfaces.maxima_lib (module), 107
sage.interfaces.mwrank (module), 129
sage.interfaces.octave (module), 133
sage.interfaces.r (module), 139
sage.interfaces.sage0 (module), 151
sage.interfaces.singular (module), 157
sage.interfaces.tachyon (module), 175
sage0_console() (in module sage.interfaces.sage0), 154
sage0_version() (in module sage.interfaces.sage0), 154
sage2matlab_matrix_string() (sage.interfaces.matlab.Matlab method), 95
sage2octave_matrix_string() (sage.interfaces.octave.Octave method), 136
sage_flattened_str_list() (sage.interfaces.singular.SingularElement method), 169
sage_global_ring() (sage.interfaces.singular.SingularElement method), 169

sage_matrix() (sage.interfaces.singular.SingularElement method), 170
sage_poly() (sage.interfaces.singular.SingularElement method), 170
sage_polystring() (sage.interfaces.singular.SingularElement method), 171
sage_rat() (in module sage.interfaces.maxima_lib), 119
sage_structured_str_list() (sage.interfaces.singular.SingularElement method), 172
SageElement (class in sage.interfaces.sage0), 154
SageFunction (class in sage.interfaces.sage0), 154
save_workspace() (sage.interfaces.gap.Gap method), 28
set() (sage.interfaces.axiom.PanAxiom method), 10
set() (sage.interfaces.gap.Gap method), 28
set() (sage.interfaces.gp.Gp method), 52
set() (sage.interfaces.kash.Kash method), 65
set() (sage.interfaces.magma.Magma method), 77
set() (sage.interfaces.maple.Maple method), 89
set() (sage.interfaces.mathematica.Mathematica method), 127
set() (sage.interfaces.matlab.Matlab method), 96
set() (sage.interfaces.matlab.MatlabElement method), 96
set() (sage.interfaces.maxima.Maxima method), 104
set() (sage.interfaces.maxima_lib.MaximaLib method), 109
set() (sage.interfaces.octave.Octave method), 137
set() (sage.interfaces.r.R method), 147
set() (sage.interfaces.sage0.Sage method), 153
set() (sage.interfaces.singular.Singular method), 167
set_default() (sage.interfaces.gp.Gp method), 52
set_gap_memory_pool_size() (in module sage.interfaces.gap), 35
set_magma_attribute() (sage.interfaces.magma.MagmaElement method), 82
set_precision() (sage.interfaces.gp.Gp method), 52
set_real_precision() (sage.interfaces.gp.Gp method), 53
set_ring() (sage.interfaces.singular.Singular method), 167
set_ring() (sage.interfaces.singular.SingularElement method), 172
set_series_precision() (sage.interfaces.gp.Gp method), 53
set_verbose() (sage.interfaces.magma.Magma method), 77
setring() (sage.interfaces.singular.Singular method), 168
SetVerbose() (sage.interfaces.magma.Magma method), 71
show() (sage.interfaces.mathematica.MathematicaElement method), 127
Singular (class in sage.interfaces.singular), 161
singular_console() (in module sage.interfaces.singular), 174
singular_version() (in module sage.interfaces.singular), 174
SingularElement (class in sage.interfaces.singular), 168
SingularError, 173
SingularFunction (class in sage.interfaces.singular), 173
SingularFunctionElement (class in sage.interfaces.singular), 173
SingularGBLogPrettyPrinter (class in sage.interfaces.singular), 173
solve_linear_system() (sage.interfaces.octave.Octave method), 137
source() (sage.interfaces.maple.Maple method), 89
source() (sage.interfaces.r.R method), 147
sr_integral() (sage.interfaces.maxima_lib.MaximaLib method), 109
sr_limit() (sage.interfaces.maxima_lib.MaximaLib method), 111
sr_sum() (sage.interfaces.maxima_lib.MaximaLib method), 112
sr_tlimit() (sage.interfaces.maxima_lib.MaximaLib method), 113

sr_to_max() (in module sage.interfaces.maxima_lib), 119
stat_model() (sage.interfaces.r.RElement method), 148
stdout_to_string() (in module sage.interfaces.maxima_lib), 120
StdOutContext (class in sage.interfaces.expect), 5
str() (sage.interfaces.gap.GapElement method), 28
str() (sage.interfaces.mathematica.MathematicaElement method), 127
string() (sage.interfaces.singular.Singular method), 168
strip_answer() (sage.interfaces.matlab.Matlab method), 96
sub() (sage.interfaces.magma.MagmaElement method), 82

T

TachyonRT (class in sage.interfaces.tachyon), 175
temp_project() (sage.interfaces.four_ti_2.FourTi2 method), 21
tilde() (sage.interfaces.r.RElement method), 149
time() (sage.interfaces.ecm.ECM method), 17
to_poly_solve() (sage.interfaces.maxima_lib.MaximaLibElement method), 114
trait_names() (sage.interfaces.axiom.PanAxiom method), 10
trait_names() (sage.interfaces.gap.Gap method), 28
trait_names() (sage.interfaces.gap.Gap_generic method), 31
trait_names() (sage.interfaces.gap.GapElement method), 29
trait_names() (sage.interfaces.gap3.GAP3Record method), 42
trait_names() (sage.interfaces.gp.Gp method), 53
trait_names() (sage.interfaces.gp.GpElement method), 54
trait_names() (sage.interfaces.magma.Magma method), 78
trait_names() (sage.interfaces.magma.MagmaElement method), 82
trait_names() (sage.interfaces.maple.Maple method), 90
trait_names() (sage.interfaces.maple.MapleElement method), 90
trait_names() (sage.interfaces.mathematica.Mathematica method), 127
trait_names() (sage.interfaces.r.R method), 148
trait_names() (sage.interfaces.r.RElement method), 149
trait_names() (sage.interfaces.sage0.Sage method), 154
trait_names() (sage.interfaces.singular.Singular method), 168
trait_names() (sage.interfaces.singular.SingularElement method), 172
type() (sage.interfaces.axiom.PanAxiomElement method), 11
type() (sage.interfaces.singular.SingularElement method), 173

U

unbind() (sage.interfaces.gap.Gap_generic method), 31
unparsed_input_form() (sage.interfaces.axiom.PanAxiomElement method), 11
usage() (sage.interfaces.tachyon.TachyonRT method), 176
user_dir() (sage.interfaces.expect.Expect method), 5

V

validate_mwrank_input() (in module sage.interfaces.mwrank), 130
version() (sage.interfaces.gap.Gap_generic method), 32
version() (sage.interfaces.gp.Gp method), 53
version() (sage.interfaces.kash.Kash method), 66
version() (sage.interfaces.magma.Magma method), 78
version() (sage.interfaces.matlab.Matlab method), 96
version() (sage.interfaces.octave.Octave method), 137

version() (sage.interfaces.r.R method), 148
version() (sage.interfaces.sage0.Sage method), 154
version() (sage.interfaces.singular.Singular method), 168

W

whos() (sage.interfaces.matlab.Matlab method), 96
with_package() (sage.interfaces.maple.Maple method), 90
write() (sage.interfaces.magma.MagmaGBLogPrettyPrinter method), 83
write() (sage.interfaces.singular.SingularGBLogPrettyPrinter method), 173
write_array() (sage.interfaces.four_ti_2.FourTi2 method), 22
write_matrix() (sage.interfaces.four_ti_2.FourTi2 method), 22
write_single_row() (sage.interfaces.four_ti_2.FourTi2 method), 22

Z

zsolve() (sage.interfaces.four_ti_2.FourTi2 method), 22