# Sage Reference Manual: Constants

*Release 6.3*

**The Sage Development Team**

August 11, 2014

# CONTENTS

CHAPTER

ONE

# MATHEMATICAL CONSTANTS

The following standard mathematical constants are defined in Sage, along with support for coercing them into GAP, PARI/GP, KASH, Maxima, Mathematica, Maple, Octave, and Singular:

```
sage: pi
pi
sage: e                 # base of the natural logarithm
e
sage: NaN               # Not a number
NaN
sage: golden_ratio
golden_ratio
sage: log2              # natural logarithm of the real number 2
log2
sage: euler_gamma       # Euler's gamma constant
euler_gamma
sage: catalan           # the Catalan constant
catalan
sage: khinchin          # Khinchin's constant
khinchin
sage: twinprime
twinprime
sage: mertens
mertens
sage: brun
brun
```

Support for coercion into the various systems means that if, e.g., you want to create $\pi$ in Maxima and Singular, you don't have to figure out the special notation for each system. You just type the following:

```
sage: maxima(pi)
%pi
sage: singular(pi)
pi
sage: gap(pi)
pi
sage: gp(pi)
3.141592653589793238462643383     # 32-bit
3.1415926535897932384626433832795028842   # 64-bit
sage: pari(pi)
3.14159265358979
sage: kash(pi)                    # optional
3.14159265358979323846264338328
sage: mathematica(pi)             # optional
Pi
```

```
sage: maple(pi)                        # optional
Pi
sage: octave(pi)                       # optional
3.14159
```

Arithmetic operations with constants also yield constants, which can be coerced into other systems or evaluated.

```
sage: a = pi + e*4/5; a
pi + 4/5*e
sage: maxima(a)
%pi+4*%e/5
sage: RealField(15)(a)           # 15 *bits* of precision
5.316
sage: gp(a)
5.316218116357029426750873360               # 32-bit
5.3162181163570294267508733603616328824     # 64-bit
sage: print mathematica(a)                     # optional
 4 E
 --- + Pi
  5
```

EXAMPLES: Decimal expansions of constants

We can obtain floating point approximations to each of these constants by coercing into the real field with given precision. For example, to 200 decimal places we have the following:

```
sage: R = RealField(200); R
Real Field with 200 bits of precision


sage: R(pi)
3.1415926535897932384626433832795028841971693993751058209749


sage: R(e)
2.7182818284590452353602874713526624977572470936999595749670


sage: R(NaN)
NaN


sage: R(golden_ratio)
1.6180339887498948482045868343656381177203091798057628621354


sage: R(log2)
0.69314718055994530941723212145817656807550013436025525412068


sage: R(euler_gamma)
0.57721566490153286060651209008240243104215933593992359880577


sage: R(catalan)
0.91596559417721901505460351493238411077414937428167213426650


sage: R(khinchin)
2.6854520010653064453097148354817956938203822939944629530512
```

EXAMPLES: Arithmetic with constants

```
sage: f = I*(e+1); f
I*e + I
sage: f^2
(I*e + I)^2
sage: _.expand()
-e^2 - 2*e - 1
```

```
sage: pp = pi+pi; pp
2*pi
sage: R(pp)
6.2831853071795864769252867665590057683943387987502116419499
```

```
sage: s = (1 + e^pi); s
e^pi + 1
sage: R(s)
24.140692632779269005729086367948547380266106242600211993445
sage: R(s-1)
23.140692632779269005729086367948547380266106242600211993445
```

```
sage: l = (1-log2)/(1+log2); l
-(log2 - 1)/(log2 + 1)
sage: R(l)
0.18123221829928249948761381864650311423330609774776013488056
```

```
sage: pim = maxima(pi)
sage: maxima.eval('fpprec : 100')
'100'
sage: pim.bfloat()
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706
```

AUTHORS:

- Alex Clemesha (2006-01-15)
- William Stein
- Alex Clemesha, William Stein (2006-02-20): added new constants; removed todos
- Didier Deshommes (2007-03-27): added constants from RQDF (deprecated)

TESTS:

Coercing the sum of a bunch of the constants to many different floating point rings:

```
sage: a = pi + e + golden_ratio + log2 + euler_gamma + catalan + khinchin + twinprime + mertens; a
mertens + twinprime + khinchin + log2 + golden_ratio + catalan + euler_gamma + pi + e
sage: parent(a)
Symbolic Ring
sage: RR(a) #abstol 1e11
13.2713479401972
sage: RealField(212)(a)
13.2713479401972493100988191995758139408711068200030748178329712
sage: RealField(230)(a)
13.27134794019724931009881919957581394087110682000307481783297118955
sage: CC(a) #abstol 1e11
13.2713479401972
sage: CDF(a)
13.2713479402
sage: ComplexField(230)(a)
```

```
13.271347940197249310098819199575813940871106820003074817832971189555
sage: RDF(a)
13.2713479402
```

Test if #8237 is fixed:

```
sage: maxima('infinity').sage()
Infinity
sage: maxima('inf').sage()
+Infinity
sage: maxima('minf').sage()
-Infinity
```

**class** sage.symbolic.constants.**Brun**(*name='brun'*)

Bases: sage.symbolic.constants.LimitedPrecisionConstant

Brun's constant is the sum of reciprocals of odd twin primes.

It is not known to very high precision; calculating the number using twin primes up to $10^{16}$ (Sebah 2002) gives the number 1.9021605831040.

EXAMPLES:

```
sage: float(brun)
Traceback (most recent call last):
...
NotImplementedError: brun is only available up to 41 bits
sage: R = RealField(41); R
Real Field with 41 bits of precision
sage: R(brun)
1.90216058310
```

**class** sage.symbolic.constants.**Catalan**(*name='catalan'*)

Bases: sage.symbolic.constants.Constant

A number appearing in combinatorics defined as the Dirichlet beta function evaluated at the number 2.

EXAMPLES:

```
sage: catalan^2 + mertens
mertens + catalan^2
```

**class** sage.symbolic.constants.**Constant**(*name*, *conversions=None*, *latex=None*, *mathml=''*, *domain='complex'*)

Bases: object

EXAMPLES:

```
sage: from sage.symbolic.constants import Constant
sage: p = Constant('p')
sage: loads(dumps(p))
p
```

**domain**()

Returns the domain of this constant. This is either positive, real, or complex, and is used by Pynac to make inferences about expressions containing this constant.

EXAMPLES:

```
sage: p = pi.pyobject(); p
pi
sage: type(_)
```

```
<class 'sage.symbolic.constants.Pi'>
sage: p.domain()
'positive'
```

**expression**()

Returns an expression for this constant.

EXAMPLES:

```
sage: a = pi.pyobject()
sage: pi2 = a.expression()
sage: pi2
pi
sage: pi2 + 2
pi + 2
sage: pi - pi2
0
```

**name**()

Returns the name of this constant.

EXAMPLES:

```
sage: from sage.symbolic.constants import Constant
sage: c = Constant('c')
sage: c.name()
'c'
```

**class** sage.symbolic.constants.**EulerGamma**(*name='euler_gamma'*)

Bases: sage.symbolic.constants.Constant

The limiting difference between the harmonic series and the natural logarithm.

EXAMPLES:

```
sage: R = RealField()
sage: R(euler_gamma)
0.577215664901533
sage: R = RealField(200); R
Real Field with 200 bits of precision
sage: R(euler_gamma)
0.57721566490153286060651209008240243104215933593992359880577
sage: eg = euler_gamma + euler_gamma; eg
2*euler_gamma
sage: R(eg)
1.1544313298030657212130241801648048620843186718798471976115
```

**class** sage.symbolic.constants.**Glaisher**(*name='glaisher'*)

Bases: sage.symbolic.constants.Constant

The Glaisher-Kinkelin constant $A = \exp(\frac{1}{12} - \zeta'(-1))$.

EXAMPLES:

```
sage: float(glaisher)
1.2824271291006226
sage: glaisher.n(digits=60)
1.28242712910062263687534256886979172776768892732500119206374
sage: a = glaisher + 2
sage: a
glaisher + 2
```

```
sage: parent(a)
Symbolic Ring
```

**class** `sage.symbolic.constants.`**`GoldenRatio`**`(`*name='golden_ratio'*`)`

Bases: `sage.symbolic.constants.Constant`

The number (1+sqrt(5))/2

EXAMPLES:

```
sage: gr = golden_ratio
sage: RR(gr)
1.61803398874989
sage: R = RealField(200)
sage: R(gr)
1.6180339887498948482045868343656381177203091798057628621354
sage: grm = maxima(golden_ratio);grm
(sqrt(5)+1)/2
sage: grm + grm
sqrt(5)+1
sage: float(grm + grm)
3.23606797749979
```

**`minpoly`**`(`*bits=None*, *degree=None*, *epsilon=0*`)`

EXAMPLES:

```
sage: golden_ratio.minpoly()
x^2 - x - 1
```

**class** `sage.symbolic.constants.`**`Khinchin`**`(`*name='khinchin'*`)`

Bases: `sage.symbolic.constants.Constant`

The geometric mean of the continued fraction expansion of any (almost any) real number.

EXAMPLES:

```
sage: float(khinchin)
2.6854520010653062
sage: khinchin.n(digits=60)
2.68545200106530644530971483548179569382038229399446295305115
sage: m = mathematica(khinchin); m              # optional
Khinchin
sage: m.N(200)                                  # optional
2.685452001065306445309714835481795693820382293994462953051152345557218859537152002801141174931
2.685452001065306445309714835481795693820382293994462953051152345557218859537152002801141174931
```

**class** `sage.symbolic.constants.`**`LimitedPrecisionConstant`**`(`*name*, *value*, ***kwds*`)`

Bases: `sage.symbolic.constants.Constant`

A class for constants that are only known to a limited precision.

EXAMPLES:

```
sage: from sage.symbolic.constants import LimitedPrecisionConstant
sage: a = LimitedPrecisionConstant('a', '1.2345678910111213l4').expression(); a
a
sage: RDF(a)
1.23456789101
sage: RealField(200)(a)
Traceback (most recent call last):
...
NotImplementedError: a is only available up to 59 bits
```

**class** sage.symbolic.constants.**Log2**(*name='log2'*)

Bases: sage.symbolic.constants.Constant

The natural logarithm of the real number 2.

EXAMPLES:

```
sage: log2
log2
sage: float(log2)
0.6931471805599453
sage: RR(log2)
0.693147180559945
sage: R = RealField(200); R
Real Field with 200 bits of precision
sage: R(log2)
0.69314718055994530941723212145817656807550013436025525412068
sage: l = (1-log2)/(1+log2); l
-(log2 - 1)/(log2 + 1)
sage: R(l)
0.18123221829928249948761381864650311423330609774776013488056
sage: maxima(log2)
log(2)
sage: maxima(log2).float()
0.6931471805599453
sage: gp(log2)
0.6931471805599453094172321215             # 32-bit
0.69314718055994530941723212145817656808   # 64-bit
sage: RealField(150)(2).log()
0.69314718055994530941723212145817656807550013
```

**class** sage.symbolic.constants.**Mertens**(*name='mertens'*)

Bases: sage.symbolic.constants.Constant

The Mertens constant is related to the Twin Primes constant and appears in Mertens' second theorem.

EXAMPLES:

```
sage: float(mertens)
0.26149721284764277
sage: mertens.n(digits=60)
0.261497212847642783755426838608695859051566648261199206192064
```

**class** sage.symbolic.constants.**NotANumber**(*name='NaN'*)

Bases: sage.symbolic.constants.Constant

Not a Number

**class** sage.symbolic.constants.**Pi**(*name='pi'*)

Bases: sage.symbolic.constants.Constant

TESTS:

```
sage: pi._latex_()
'\pi'
sage: latex(pi)
\pi
sage: mathml(pi)
<mi>&pi;</mi>
```

**class** sage.symbolic.constants.**TwinPrime**(*name='twinprime'*)

Bases: sage.symbolic.constants.Constant

The Twin Primes constant is defined as $\prod 1 - 1/(p-1)^2$ for primes $p > 2$.

EXAMPLES:

```
sage: float(twinprime)
0.6601618158468696
sage: twinprime.n(digits=60)
0.660161815846869573927812110014555778432623360284733413319448
```

`sage.symbolic.constants.`**`unpickle_Constant`**(*class_name*, *name*, *conversions*, *latex*, *mathml*, *domain*)

EXAMPLES:

```
sage: from sage.symbolic.constants import unpickle_Constant
sage: a = unpickle_Constant('Constant', 'a', {}, 'aa', '', 'positive')
sage: a.domain()
'positive'
sage: latex(a)
aa
```

Note that if the name already appears in the `constants_name_table`, then that will be returned instead of constructing a new object:

```
sage: pi = unpickle_Constant('Pi', 'pi', None, None, None, None)
sage: pi._maxima_init_()
'%pi'
```

CHAPTER

TWO

# WRAPPER AROUND PYNAC'S CONSTANTS

Wrapper around Pynac's constants

**class** `sage.symbolic.constants_c.E`

Bases: `sage.symbolic.expression.Expression`

Dummy class to represent base of the natural logarithm.

The base of the natural logarithm `e` is not a constant in GiNaC/Sage. It is represented by `exp(1)`.

This class provides a dummy object that behaves well under addition, multiplication, etc. and on exponentiation calls the function `exp`.

EXAMPLES:

The constant defined at the top level is just `exp(1)`:

```
sage: e.operator()
exp
sage: e.operands()
[1]
```

Arithmetic works:

```
sage: e + 2
e + 2
sage: 2 + e
e + 2
sage: 2*e
2*e
sage: e*2
2*e
sage: x*e
x*e
sage: var('a,b')
(a, b)
sage: t = e^(a+b); t
e^(a + b)
sage: t.operands()
[a + b]
```

Numeric evaluation, conversion to other systems, and pickling works as expected. Note that these are properties of the `exp()` function, not this class:

```
sage: RR(e)
2.71828182845905
sage: R = RealField(200); R
Real Field with 200 bits of precision
sage: R(e)
2.7182818284590452353602874713526624977572470936999595749670
sage: em = 1 + e^(1-e); em
e^(-e + 1) + 1
sage: R(em)
1.1793740787340171819619895873183164984596816017589156131574
sage: maxima(e).float()
2.718281828459045
sage: t = mathematica(e)                # optional
sage: t                                 # optional
E
sage: float(t)                          # optional
2.718281828459045...

sage: loads(dumps(e))
e

sage: float(e)
2.718281828459045...
sage: e.__float__()
2.718281828459045...
sage: e._mpfr_(RealField(100))
2.7182818284590452353602874714
sage: e._real_double_(RDF)
2.71828182846
sage: import sympy
sage: sympy.E == e # indirect doctest
True
```

TESTS:

```
sage: t = e^a; t
e^a
sage: t^b
(e^a)^b
sage: SR(1).exp()
e
```

Testing that it works with matrices (see trac ticket #4735):

```
sage: m = matrix(QQ, 2, 2, [1,0,0,1])
sage: e^m
[e 0]
[0 e]
```

**class** `sage.symbolic.constants_c.`**`PynacConstant`**

Bases: `object`

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

**`expression`**()

Returns this constant as an Expression.

EXAMPLES:

```
sage: from sage.symbolic.constants_c import PynacConstant
sage: f = PynacConstant('foo', 'foo', 'real')
sage: f + 2
Traceback (most recent call last):
...
TypeError: unsupported operand parent(s) for '+': '<type 'sage.symbolic.constants_c.PynacCon

sage: foo = f.expression(); foo
foo
sage: foo + 2
foo + 2
```

**name**()

Returns the name of this constant.

EXAMPLES:

```
sage: from sage.symbolic.constants_c import PynacConstant
sage: f = PynacConstant('foo', 'foo', 'real')
sage: f.name()
'foo'
```

**serial**()

Returns the underlying Pynac serial for this constant.

EXAMPLES:

```
sage: from sage.symbolic.constants_c import PynacConstant
sage: f = PynacConstant('foo', 'foo', 'real')
sage: f.serial()   #random
15
```

CHAPTER

THREE

# INDICES AND TABLES

- Index
- Module Index
- Search Page

# PYTHON MODULE INDEX

### S

# INDEX

## P



## S



## T



## U