Constants

Release 10.6

The Sage Development Team

Apr 01, 2025

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
```

```
>>> from sage.all import *
>>> pi
pi
>>> e             # base of the natural logarithm
e
>>> NaN           # Not a number
NaN
>>> golden_ratio
golden_ratio
>>> log2          # natural logarithm of the real number 2
log2
>>> euler_gamma   # Euler's gamma constant
euler_gamma
>>> catalan       # the Catalan constant
catalan
>>> khinchin      # Khinchin's constant
```

(continues on next page)

(continued from previous page)

```
khinchin
>>> twinprime
twinprime
>>> mertens
mertens
```

Support for coercion into the various systems means that if, e.g., you want to create π 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)
3.141592653589793238462643383279502884
sage: pari(pi)
3.14159265358979
sage: kash(pi)                          # optional - kash
3.14159265358979323846264338328
sage: mathematica(pi)                   # optional - mathematica
Pi
sage: pi._maple_init_()
'Pi'
sage: octave(pi)                        # optional - octave
3.14159
```

```
>>> from sage.all import *
>>> maxima(pi)
%pi
>>> singular(pi)
pi
>>> gap(pi)
pi
>>> gp(pi)
3.141592653589793238462643383279502884
>>> pari(pi)
3.14159265358979
>>> kash(pi)                          # optional - kash
3.14159265358979323846264338328
>>> mathematica(pi)                   # optional - mathematica
Pi
>>> pi._maple_init_()
'Pi'
>>> octave(pi)                        # optional - octave
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
```

(continues on next page)

(continued from previous page)

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

```
>>> from sage.all import *
>>> a = pi + e*Integer(4)/Integer(5); a
pi + 4/5*e
>>> maxima(a)
%pi+(4*%e)/5
>>> RealField(Integer(15))(a)           # 15 *bits* of precision
5.316
>>> gp(a)
5.3162181163570294267508733603616328824
>>> print(mathematica(a))                   # optional - mathematica
 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 binary places we have the following:

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

```
>>> from sage.all import *
>>> R = RealField(Integer(200)); R
Real Field with 200 bits of precision
```

```
sage: R(pi)
3.1415926535897932384626433832795028841971693993751058209749
```

```
>>> from sage.all import *
>>> R(pi)
3.1415926535897932384626433832795028841971693993751058209749
```

```
sage: R(e)
2.7182818284590452353602874713526624977572470936999595749670
```

```
>>> from sage.all import *
>>> R(e)
2.7182818284590452353602874713526624977572470936999595749670
```

```
sage: R(NaN)
NaN
```

```
>>> from sage.all import *
>>> R(NaN)
NaN
```

```
sage: R(golden_ratio)
1.6180339887498948482045868343656381177203091798057628621354
```

```
>>> from sage.all import *
>>> R(golden_ratio)
1.6180339887498948482045868343656381177203091798057628621354
```

```
sage: R(log2)
0.69314718055994530941723212145817656807550013436025525412068
```

```
>>> from sage.all import *
>>> R(log2)
0.69314718055994530941723212145817656807550013436025525412068
```

```
sage: R(euler_gamma)
0.57721566490153286060651209008240243104215933593992359880577
```

```
>>> from sage.all import *
>>> R(euler_gamma)
0.57721566490153286060651209008240243104215933593992359880577
```

```
sage: R(catalan)
0.91596559417721901505460351493238411077414937428167213426650
```

```
>>> from sage.all import *
>>> R(catalan)
0.91596559417721901505460351493238411077414937428167213426650
```

```
sage: R(khinchin)
2.6854520010653064453097148354817956938203822939944629530512
```

```
>>> from sage.all import *
>>> 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
```

```
>>> from sage.all import *
>>> f = I*(e+Integer(1)); f
I*e + I
>>> f**Integer(2)
(I*e + I)^2
>>> _.expand()
-e^2 - 2*e - 1
```

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

```
>>> from sage.all import *
>>> pp = pi+pi; pp
2*pi
>>> R(pp)
6.2831853071795864769252867665590057683943387987502116419499
```

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

```
>>> from sage.all import *
>>> s = (Integer(1) + e**pi); s
e^pi + 1
>>> R(s)
24.140692632779269005729086367948547380266106242600211993445
>>> R(s-Integer(1))
23.140692632779269005729086367948547380266106242600211993445
```

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

```
>>> from sage.all import *
>>> l = (Integer(1)-log2)/(Integer(1)+log2); l
-(log2 - 1)/(log2 + 1)
>>> R(l)
0.18123221829928249948761381864650311423330609774776013488056
```

```
sage: pim = maxima(pi)
sage: maxima.eval('fpprec : 100')
'100'
sage: pim.bfloat()
3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628
↪034825342117068b0
```

```
>>> from sage.all import *
>>> pim = maxima(pi)
>>> maxima.eval('fpprec : 100')
'100'
>>> pim.bfloat()
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862
↪034825342117068b0
```

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)

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

Bases: `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
```

```
>>> from sage.all import *
>>> catalan**Integer(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
```

```
>>> from sage.all import *
>>> from sage.symbolic.constants import Constant
>>> p = Constant('p')
>>> loads(dumps(p))
p
```

`domain`()

Return 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'
```

```
>>> from sage.all import *
>>> p = pi.pyobject(); p
pi
>>> type(_)
<class 'sage.symbolic.constants.Pi'>
>>> p.domain()
'positive'
```

expression()

Return 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
```

```
>>> from sage.all import *
>>> a = pi.pyobject()
>>> pi2 = a.expression()
>>> pi2
pi
>>> pi2 + Integer(2)
pi + 2
>>> pi - pi2
0
```

name()

Return the name of this constant.

EXAMPLES:

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

```
>>> from sage.all import *
>>> from sage.symbolic.constants import Constant
>>> c = Constant('c')
```

(continues on next page)

(continued from previous page)

```
>>> c.name()
'c'
```

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

Bases: 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
```

```
>>> from sage.all import *
>>> R = RealField()
>>> R(euler_gamma)
0.577215664901533
>>> R = RealField(Integer(200)); R
Real Field with 200 bits of precision
>>> R(euler_gamma)
0.57721566490153286060651209008240243104215933593992359880577
>>> eg = euler_gamma + euler_gamma; eg
2*euler_gamma
>>> R(eg)
1.1544313298030657212130241801648048620843186718798471976115
```

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

Bases: 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
```

```
>>> from sage.all import *
>>> float(glaisher)
1.2824271291006226
>>> glaisher.n(digits=Integer(60))
1.28242712910062263687534256886979172776768892732500119206374
>>> a = glaisher + Integer(2)
>>> a
glaisher + 2
>>> parent(a)
Symbolic Ring
```

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

Bases: *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
```

```
>>> from sage.all import *
>>> gr = golden_ratio
>>> RR(gr)
1.61803398874989
>>> R = RealField(Integer(200))
>>> R(gr)
1.6180339887498948482045868343656381177203091798057628621354
>>> grm = maxima(golden_ratio);grm
(sqrt(5)+1)/2
>>> grm + grm
sqrt(5)+1
>>> float(grm + grm)
3.23606797749979
```

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

EXAMPLES:

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

```
>>> from sage.all import *
>>> golden_ratio.minpoly()
```

(continues on next page)

(continued from previous page)

```
x^2 - x - 1
```

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

Bases: 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 - mathematica
Khinchin
sage: m.N(200)                                  # optional - mathematica
2.685452001065306445309714835481795693820382293...32852204481940961807
```

```
>>> from sage.all import *
>>> float(khinchin)
2.6854520010653062
>>> khinchin.n(digits=Integer(60))
2.68545200106530644530971483548179569382038229399446295305115
>>> m = mathematica(khinchin); m              # optional - mathematica
Khinchin
>>> m.N(Integer(200))                                  # optional - mathematica
2.685452001065306445309714835481795693820382293...32852204481940961807
```

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

Bases: 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)
```

(continues on next page)

(continued from previous page)

```
0.69314718055994530941723212145817656807
sage: RealField(150)(2).log()
0.69314718055994530941723212145817656807550013
```

```
>>> from sage.all import *
>>> log2
log2
>>> float(log2)
0.6931471805599453
>>> RR(log2)
0.693147180559945
>>> R = RealField(Integer(200)); R
Real Field with 200 bits of precision
>>> R(log2)
0.69314718055994530941723212145817656807550013436025525412068
>>> l = (Integer(1)-log2)/(Integer(1)+log2); l
-(log2 - 1)/(log2 + 1)
>>> R(l)
0.18123221829928249948761381864650311423330609774776013488056
>>> maxima(log2)
log(2)
>>> maxima(log2).float()
0.6931471805599453
>>> gp(log2)
0.69314718055994530941723212145817656807
>>> RealField(Integer(150))(Integer(2)).log()
0.69314718055994530941723212145817656807550013
```

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

Bases: *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
```

```
>>> from sage.all import *
>>> float(mertens)
0.26149721284764277
>>> mertens.n(digits=Integer(60))
0.261497212847642783755426838608695859051566648261199206192064
```

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

Bases: *Constant*

Not a Number

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

Bases: *Constant*

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

Bases: `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
```

```
>>> from sage.all import *
>>> float(twinprime)
0.6601618158468696
>>> twinprime.n(digits=Integer(60))
0.660161815846869573927812110014555778432623360284733413319448
```

sage.symbolic.constants.**pi = pi**

The formal square root of -1.

EXAMPLES:

```
sage: SR.I()
I
sage: SR.I()^2
-1
```

```
>>> from sage.all import *
>>> SR.I()
I
>>> SR.I()**Integer(2)
-1
```

Note that conversions to real fields will give TypeErrors:

```
sage: float(SR.I())
Traceback (most recent call last):
...
TypeError: unable to simplify to float approximation
sage: gp(SR.I())
I
sage: RR(SR.I())
Traceback (most recent call last):
...
TypeError: unable to convert '1.00000000000000*I' to a real number
```

```
>>> from sage.all import *
>>> float(SR.I())
Traceback (most recent call last):
...
TypeError: unable to simplify to float approximation
>>> gp(SR.I())
I
```

(continues on next page)

(continued from previous page)

```
>>> RR(SR.I())
Traceback (most recent call last):
...
TypeError: unable to convert '1.00000000000000*I' to a real number
```

Expressions involving I that are real-valued can be converted to real fields:

```
sage: float(I*I)
-1.0
sage: RR(I*I)
-1.00000000000000
```

```
>>> from sage.all import *
>>> float(I*I)
-1.0
>>> RR(I*I)
-1.00000000000000
```

We can convert to complex fields:

```
sage: C = ComplexField(200); C
Complex Field with 200 bits of precision
sage: C(SR.I())
1.0000000000000000000000000000000000000000000000000000000000*I
sage: SR.I()._complex_mpfr_field_(ComplexField(53))
1.00000000000000*I

sage: SR.I()._complex_double_(CDF)
1.0*I
sage: CDF(SR.I())
1.0*I

sage: z = SR.I() + I; z
2*I
sage: C(z)
2.0000000000000000000000000000000000000000000000000000000000*I
sage: 1e8*SR.I()
1.00000000000000e8*I

sage: complex(SR.I())
1j

sage: QQbar(SR.I())
I

sage: abs(SR.I())
1

sage: SR.I().minpoly()
x^2 + 1
sage: maxima(2*SR.I())
2*%i
```

```
>>> from sage.all import *
>>> C = ComplexField(Integer(200)); C
Complex Field with 200 bits of precision
>>> C(SR.I())
1.0000000000000000000000000000000000000000000000000000000000*I
>>> SR.I()._complex_mpfr_field_(ComplexField(Integer(53)))
1.00000000000000*I

>>> SR.I()._complex_double_(CDF)
1.0*I
>>> CDF(SR.I())
1.0*I

>>> z = SR.I() + I; z
2*I
>>> C(z)
2.0000000000000000000000000000000000000000000000000000000000*I
>>> RealNumber('1e8')*SR.I()
1.00000000000000e8*I

>>> complex(SR.I())
1j

>>> QQbar(SR.I())
I

>>> abs(SR.I())
1

>>> SR.I().minpoly()
x^2 + 1
>>> maxima(Integer(2)*SR.I())
2*%i
```

`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
```

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

```
>>> from sage.all import *
>>> pi = unpickle_Constant('Pi', 'pi', None, None, None, None)
>>> pi._maxima_init_()
'%pi'
```

CHAPTER

TWO

INDICES AND TABLES

- Index
- Module Index
- Search Page

PYTHON MODULE INDEX

INDEX