rem **(page 2517)**
Analogue of `a % b`

div **(page 2509)**
Analogue of `divmod(a, b)`

exquo **(page 2511)**
Analogue of `a / b`

rem(*a*, *b*)

Modulo division of *a* and *b*. Analogue of `a % b`.

`K.rem(a, b)` is equivalent to `K.div(a, b)[1]`. See *div()* (page 2509) for more explanation.

See also:

quo **(page 2516)**
Analogue of `a // b`

div **(page 2509)**
Analogue of `divmod(a, b)`

exquo **(page 2511)**
Analogue of `a / b`

revert(*a*)

Returns `a**(-1)` if possible.

sqrt(*a*)

Returns square root of `a`.

sub(*a*, *b*)

Difference of `a` and `b`, implies `__sub__`.

to_sympy(*a*)

Convert domain element *a* to a SymPy expression (Expr).

Parameters

a: domain element

An element of this *Domain* (page 2504).

Returns

expr: Expr

A normal SymPy expression of type *Expr* (page 947).

Explanation

Convert a *Domain* (page 2504) element *a* to *Expr* (page 947). Most public SymPy functions work with objects of type *Expr* (page 947). The elements of a *Domain* (page 2504) have a different internal representation. It is not possible to mix domain elements with *Expr* (page 947) so each domain has *to_sympy()* (page 2517) and *from_sympy()* (page 2513) methods to convert its domain elements to and from *Expr* (page 947).

Examples

Construct an element of the *QQ* (page 2529) domain and then convert it to *Expr* (page 947).

```
>>> from sympy import QQ, Expr
>>> q_domain = QQ(2)
>>> q_domain
2
>>> q_expr = QQ.to_sympy(q_domain)
>>> q_expr
2
```

Although the printed forms look similar these objects are not of the same type.

```
>>> isinstance(q_domain, Expr)
False
>>> isinstance(q_expr, Expr)
True
```

Construct an element of *K[x]* (page 2547) and convert to *Expr* (page 947).

```
>>> from sympy import Symbol
>>> x = Symbol('x')
>>> K = QQ[x]
>>> x_domain = K.gens[0]  # generator x as a domain element
>>> p_domain = x_domain**2/3 + 1
>>> p_domain
1/3*x**2 + 1
>>> p_expr = K.to_sympy(p_domain)
>>> p_expr
x**2/3 + 1
```

The *from_sympy()* (page 2513) method is used for the opposite conversion from a normal SymPy expression to a domain element.

```
>>> p_domain == p_expr
False
>>> K.from_sympy(p_expr) == p_domain
True
>>> K.to_sympy(p_domain) == p_expr
True
>>> K.from_sympy(K.to_sympy(p_domain)) == p_domain
True
>>> K.to_sympy(K.from_sympy(p_expr)) == p_expr
True
```

The *from_sympy()* (page 2513) method makes it easier to construct domain elements interactively.

```
>>> from sympy import Symbol
>>> x = Symbol('x')
>>> K = QQ[x]
```

(continues on next page)

(continued from previous page)

```
>>> K.from_sympy(x**2/3 + 1)
1/3*x**2 + 1
```

See also:

from_sympy (page 2513), *convert_from* (page 2508)

property tp

Alias for *dtype* (page 2510)

unify(*K1*, *symbols=None*)

Construct a minimal domain that contains elements of `K0` and `K1`.

Known domains (from smallest to largest):

- `GF(p)`
- `ZZ`
- `QQ`
- `RR(prec, tol)`
- `CC(prec, tol)`
- `ALG(a, b, c)`
- `K[x, y, z]`
- `K(x, y, z)`
- `EX`

zero: Optional[Any] = None

The zero element of the *Domain* (page 2504):

```
>>> from sympy import QQ
>>> QQ.zero
0
>>> QQ.of_type(QQ.zero)
True
```

See also:

of_type (page 2516), *one* (page 2516)

class sympy.polys.domains.domainelement.**DomainElement**

Represents an element of a domain.

Mix in this trait into a class whose instances should be recognized as elements of a domain. Method `parent()` gives that domain.

parent()

Get the domain associated with `self`

Examples

```
>>> from sympy import ZZ, symbols
>>> x, y = symbols('x, y')
>>> K = ZZ[x,y]
>>> p = K(x)**2 + K(y)**2
>>> p
x**2 + y**2
>>> p.parent()
ZZ[x,y]
```

Notes

This is used by *convert()* (page 2508) to identify the domain associated with a domain element.

class sympy.polys.domains.field.**Field**

Represents a field domain.

div(*a*, *b*)

Division of a and b, implies `__truediv__`.

exquo(*a*, *b*)

Exact quotient of a and b, implies `__truediv__`.

gcd(*a*, *b*)

Returns GCD of a and b.

This definition of GCD over fields allows to clear denominators in $primitive()$.

Examples

```
>>> from sympy.polys.domains import QQ
>>> from sympy import S, gcd, primitive
>>> from sympy.abc import x
```

```
>>> QQ.gcd(QQ(2, 3), QQ(4, 9))
2/9
>>> gcd(S(2)/3, S(4)/9)
2/9
>>> primitive(2*x/3 + S(4)/9)
(2/9, 3*x + 2)
```

get_field()

Returns a field associated with self.

get_ring()

Returns a ring associated with self.

is_unit(*a*)

Return true if a is a invertible

lcm(*a*, *b*)

Returns LCM of a and b.

```
>>> from sympy.polys.domains import QQ
>>> from sympy import S, lcm
```

```
>>> QQ.lcm(QQ(2, 3), QQ(4, 9))
4/3
>>> lcm(S(2)/3, S(4)/9)
4/3
```

quo(*a*, *b*)

Quotient of a and b, implies `__truediv__`.

rem(*a*, *b*)

Remainder of a and b, implies nothing.

revert(*a*)

Returns `a**(-1)` if possible.

class sympy.polys.domains.ring.**Ring**

Represents a ring domain.

denom(*a*)

Returns denominator of a.

div(*a*, *b*)

Division of a and b, implies `__divmod__`.

exquo(*a*, *b*)

Exact quotient of a and b, implies `__floordiv__`.

free_module(*rank*)

Generate a free module of rank `rank` over self.

```
>>> from sympy.abc import x
>>> from sympy import QQ
>>> QQ.old_poly_ring(x).free_module(2)
QQ[x]**2
```

get_ring()

Returns a ring associated with `self`.

ideal(**gens*)

Generate an ideal of `self`.

```
>>> from sympy.abc import x
>>> from sympy import QQ
>>> QQ.old_poly_ring(x).ideal(x**2)
<x**2>
```

invert(*a*, *b*)

Returns inversion of `a mod b`.

numer(*a*)

Returns numerator of a.

quo(*a*, *b*)
Quotient of a and b, implies `__floordiv__`.

quotient_ring(*e*)
Form a quotient ring of `self`.

Here `e` can be an ideal or an iterable.

```
>>> from sympy.abc import x
>>> from sympy import QQ
>>> QQ.old_poly_ring(x).quotient_ring(QQ.old_poly_ring(x).ideal(x**2))
QQ[x]/<x**2>
>>> QQ.old_poly_ring(x).quotient_ring([x**2])
QQ[x]/<x**2>
```

The division operator has been overloaded for this:

```
>>> QQ.old_poly_ring(x)/[x**2]
QQ[x]/<x**2>
```

rem(*a*, *b*)
Remainder of a and b, implies `__mod__`.

revert(*a*)
Returns `a**(-1)` if possible.

class `sympy.polys.domains.simpledomain.`**SimpleDomain**
Base class for simple domains, e.g. ZZ, QQ.

inject(**gens*)
Inject generators into this domain.

class `sympy.polys.domains.compositedomain.`**CompositeDomain**
Base class for composite domains, e.g. ZZ[x], ZZ(X).

drop(**symbols*)
Drop generators from this domain.

inject(**symbols*)
Inject generators into this domain.

GF(p)

class `sympy.polys.domains.`**FiniteField**(*mod*, *symmetric=True*)
Finite field of prime order *GF(p)* (page 2522)

A *GF(p)* (page 2522) domain represents a finite field $\mathbb{F}_p$ of prime order as *Domain* (page 2504) in the domain system (see *Introducing the Domains of the poly module* (page 2477)).

A *Poly* (page 2378) created from an expression with integer coefficients will have the domain *ZZ* (page 2525). However, if the `modulus=p` option is given then the domain will be a finite field instead.

```
>>> from sympy import Poly, Symbol
>>> x = Symbol('x')
>>> p = Poly(x**2 + 1)
>>> p
Poly(x**2 + 1, x, domain='ZZ')
>>> p.domain
ZZ
>>> p2 = Poly(x**2 + 1, modulus=2)
>>> p2
Poly(x**2 + 1, x, modulus=2)
>>> p2.domain
GF(2)
```

It is possible to factorise a polynomial over *GF(p)* (page 2522) using the modulus argument to `factor()` (page 2373) or by specifying the domain explicitly. The domain can also be given as a string.

```
>>> from sympy import factor, GF
>>> factor(x**2 + 1)
x**2 + 1
>>> factor(x**2 + 1, modulus=2)
(x + 1)**2
>>> factor(x**2 + 1, domain=GF(2))
(x + 1)**2
>>> factor(x**2 + 1, domain='GF(2)')
(x + 1)**2
```

It is also possible to use *GF(p)* (page 2522) with the `cancel()` (page 2376) and `gcd()` (page 2368) functions.

```
>>> from sympy import cancel, gcd
>>> cancel((x**2 + 1)/(x + 1))
(x**2 + 1)/(x + 1)
>>> cancel((x**2 + 1)/(x + 1), domain=GF(2))
x + 1
>>> gcd(x**2 + 1, x + 1)
1
>>> gcd(x**2 + 1, x + 1, domain=GF(2))
x + 1
```

When using the domain directly *GF(p)* (page 2522) can be used as a constructor to create instances which then support the operations `+,-,*,**,/`

```
>>> from sympy import GF
>>> K = GF(5)
>>> K
GF(5)
>>> x = K(3)
>>> y = K(2)
>>> x
3 mod 5
>>> y
2 mod 5
```

(continues on next page)

(continued from previous page)

```
>>> x * y
1 mod 5
>>> x / y
4 mod 5
```

Notes

It is also possible to create a *GF(p)* (page 2522) domain of **non-prime** order but the resulting ring is **not** a field: it is just the ring of the integers modulo n.

```
>>> K = GF(9)
>>> z = K(3)
>>> z
3 mod 9
>>> z**2
0 mod 9
```

It would be good to have a proper implementation of prime power fields (GF(p**n)) but these are not yet implemented in SymPY.

characteristic()

Return the characteristic of this domain.

from_FF(*a*, *K0=None*)

Convert ModularInteger(int) to dtype.

from_FF_gmpy(*a*, *K0=None*)

Convert ModularInteger(mpz) to dtype.

from_FF_python(*a*, *K0=None*)

Convert ModularInteger(int) to dtype.

from_QQ(*a*, *K0=None*)

Convert Python's Fraction to dtype.

from_QQ_gmpy(*a*, *K0=None*)

Convert GMPY's mpq to dtype.

from_QQ_python(*a*, *K0=None*)

Convert Python's Fraction to dtype.

from_RealField(*a*, *K0*)

Convert mpmath's mpf to dtype.

from_ZZ(*a*, *K0=None*)

Convert Python's int to dtype.

from_ZZ_gmpy(*a*, *K0=None*)

Convert GMPY's mpz to dtype.

from_ZZ_python(*a*, *K0=None*)

Convert Python's int to dtype.

from_sympy(*a*)
Convert SymPy's Integer to SymPy's `Integer`.

get_field()
Returns a field associated with `self`.

to_sympy(*a*)
Convert a to a SymPy object.

class sympy.polys.domains.**PythonFiniteField**(*mod*, *symmetric=True*)
Finite field based on Python's integers.

class sympy.polys.domains.**GMPYFiniteField**(*mod*, *symmetric=True*)
Finite field based on GMPY integers.

ZZ

The *ZZ* (page 2525) domain represents the integers $\mathbb{Z}$ as a *Domain* (page 2504) in the domain system (see *Introducing the Domains of the poly module* (page 2477)).

By default a *Poly* (page 2378) created from an expression with integer coefficients will have the domain *ZZ* (page 2525):

```
>>> from sympy import Poly, Symbol
>>> x = Symbol('x')
>>> p = Poly(x**2 + 1)
>>> p
Poly(x**2 + 1, x, domain='ZZ')
>>> p.domain
ZZ
```

The corresponding field of fractions is the domain of the rationals *QQ* (page 2529). Conversely *ZZ* (page 2525) is the ring of integers of *QQ* (page 2529):

```
>>> from sympy import ZZ, QQ
>>> ZZ.get_field()
QQ
>>> QQ.get_ring()
ZZ
```

When using the domain directly *ZZ* (page 2525) can be used as a constructor to create instances which then support the operations `+,-,*,**,//,%` (true division `/` should not be used with *ZZ* (page 2525) - see the *exquo()* (page 2511) domain method):

```
>>> x = ZZ(5)
>>> y = ZZ(2)
>>> x // y  # floor division
2
>>> x % y   # modulo division (remainder)
1
```

The *gcd()* (page 2514) method can be used to compute the gcd of any two elements:

```
>>> ZZ.gcd(ZZ(10), ZZ(2))
2
```

There are two implementations of *ZZ* (page 2525) in SymPy. If gmpy or gmpy2 is installed then *ZZ* (page 2525) will be implemented by *GMPYIntegerRing* (page 2528) and the elements will be instances of the gmpy.mpz type. Otherwise if gmpy and gmpy2 are not installed then *ZZ* (page 2525) will be implemented by *PythonIntegerRing* (page 2528) which uses Python's standard builtin int type. With larger integers gmpy can be more efficient so it is preferred when available.

class sympy.polys.domains.**IntegerRing**

The domain ZZ representing the integers $\mathbb{Z}$.

The *IntegerRing* (page 2526) class represents the ring of integers as a *Domain* (page 2504) in the domain system. *IntegerRing* (page 2526) is a super class of *PythonIntegerRing* (page 2528) and *GMPYIntegerRing* (page 2528) one of which will be the implementation for *ZZ* (page 2525) depending on whether or not gmpy or gmpy2 is installed.

See also:

Domain (page 2504)

algebraic_field(**extension*, *alias=None*)

Returns an algebraic field, i.e. $\mathbb{Q}(\alpha, \ldots)$.

Parameters

***extension** : One or more *Expr* (page 947).

Generators of the extension. These should be expressions that are algebraic over $\mathbb{Q}$.

alias : str, *Symbol* (page 976), None, optional (default=None)

If provided, this will be used as the alias symbol for the primitive element of the returned *AlgebraicField* (page 2539).

Returns

AlgebraicField (page 2539)

A *Domain* (page 2504) representing the algebraic field extension.

Examples

```
>>> from sympy import ZZ, sqrt
>>> ZZ.algebraic_field(sqrt(2))
QQ<sqrt(2)>
```

factorial(*a*)

Compute factorial of a.

from_AlgebraicField(*a*, *K0*)

Convert a *ANP* (page 2568) object to *ZZ* (page 2525).

See *convert()* (page 2508).

from_FF(*a*, *K0*)

Convert ModularInteger(int) to GMPY's mpz.

from_FF_gmpy(*a*, *K0*)

Convert ModularInteger(mpz) to GMPY's mpz.

from_FF_python(*a*, *K0*)
Convert `ModularInteger(int)` to GMPY's `mpz`.

from_QQ(*a*, *K0*)
Convert Python's `Fraction` to GMPY's `mpz`.

from_QQ_gmpy(*a*, *K0*)
Convert GMPY `mpq` to GMPY's `mpz`.

from_QQ_python(*a*, *K0*)
Convert Python's `Fraction` to GMPY's `mpz`.

from_RealField(*a*, *K0*)
Convert mpmath's `mpf` to GMPY's `mpz`.

from_ZZ(*a*, *K0*)
Convert Python's `int` to GMPY's `mpz`.

from_ZZ_gmpy(*a*, *K0*)
Convert GMPY's `mpz` to GMPY's `mpz`.

from_ZZ_python(*a*, *K0*)
Convert Python's `int` to GMPY's `mpz`.

from_sympy(*a*)
Convert SymPy's Integer to `dtype`.

gcd(*a*, *b*)
Compute GCD of `a` and `b`.

gcdex(*a*, *b*)
Compute extended GCD of `a` and `b`.

get_field()
Return the associated field of fractions *QQ* (page 2529)

Returns
QQ (page 2529):

The associated field of fractions *QQ* (page 2529), a *Domain* (page 2504) representing the rational numbers $\mathbb{Q}$.

Examples

```
>>> from sympy import ZZ
>>> ZZ.get_field()
QQ
```

lcm(*a*, *b*)
Compute LCM of `a` and `b`.

log(*a*, *b*)
logarithm of *a* to the base *b*

Parameters
a: number

b: number

Returns

$lfloor \log(a, b)$

$rfloor$:

Floor of the logarithm of a to the base b

Examples

```
>>> from sympy import ZZ
>>> ZZ.log(ZZ(8), ZZ(2))
3
>>> ZZ.log(ZZ(9), ZZ(2))
3
```

Notes

This function uses `math.log` which is based on `float` so it will fail for large integer arguments.

sqrt(*a*)

Compute square root of `a`.

to_sympy(*a*)

Convert `a` to a SymPy object.

class sympy.polys.domains.**PythonIntegerRing**

Integer ring based on Python's `int` type.

This will be used as *ZZ* (page 2525) if `gmpy` and `gmpy2` are not installed. Elements are instances of the standard Python `int` type.

class sympy.polys.domains.**GMPYIntegerRing**

Integer ring based on GMPY's `mpz` type.

This will be the implementation of *ZZ* (page 2525) if `gmpy` or `gmpy2` is installed. Elements will be of type `gmpy.mpz`.

factorial(*a*)

Compute factorial of `a`.

from_FF_gmpy(*a, K0*)

Convert `ModularInteger(mpz)` to GMPY's `mpz`.

from_FF_python(*a, K0*)

Convert `ModularInteger(int)` to GMPY's `mpz`.

from_QQ(*a, K0*)

Convert Python's `Fraction` to GMPY's `mpz`.

from_QQ_gmpy(*a, K0*)

Convert GMPY `mpq` to GMPY's `mpz`.

from_QQ_python(*a, K0*)

Convert Python's `Fraction` to GMPY's `mpz`.

from_RealField(*a*, *K0*)
Convert mpmath's `mpf` to GMPY's `mpz`.

from_ZZ_gmpy(*a*, *K0*)
Convert GMPY's `mpz` to GMPY's `mpz`.

from_ZZ_python(*a*, *K0*)
Convert Python's `int` to GMPY's `mpz`.

from_sympy(*a*)
Convert SymPy's Integer to `dtype`.

gcd(*a*, *b*)
Compute GCD of `a` and `b`.

gcdex(*a*, *b*)
Compute extended GCD of `a` and `b`.

lcm(*a*, *b*)
Compute LCM of `a` and `b`.

sqrt(*a*)
Compute square root of `a`.

to_sympy(*a*)
Convert `a` to a SymPy object.

QQ

The *QQ* (page 2529) domain represents the rationals $\mathbb{Q}$ as a *Domain* (page 2504) in the domain system (see *Introducing the Domains of the poly module* (page 2477)).

By default a *Poly* (page 2378) created from an expression with rational coefficients will have the domain *QQ* (page 2529):

```
>>> from sympy import Poly, Symbol
>>> x = Symbol('x')
>>> p = Poly(x**2 + x/2)
>>> p
Poly(x**2 + 1/2*x, x, domain='QQ')
>>> p.domain
QQ
```

The corresponding ring of integers is the *Domain* (page 2504) of the integers *ZZ* (page 2525). Conversely *QQ* (page 2529) is the field of fractions of *ZZ* (page 2525):

```
>>> from sympy import ZZ, QQ
>>> QQ.get_ring()
ZZ
>>> ZZ.get_field()
QQ
```

When using the domain directly *QQ* (page 2529) can be used as a constructor to create instances which then support the operations `+,-,*,**,/` (true division `/` is always possible for nonzero divisors in *QQ* (page 2529)):

```
>>> x = QQ(5)
>>> y = QQ(2)
>>> x / y  # true division
5/2
```

There are two implementations of *QQ* (page 2529) in SymPy. If gmpy or gmpy2 is installed then *QQ* (page 2529) will be implemented by *GMPYRationalField* (page 2531) and the elements will be instances of the gmpy.mpq type. Otherwise if gmpy and gmpy2 are not installed then *QQ* (page 2529) will be implemented by *PythonRationalField* (page 2531) which is a pure Python class as part of sympy. The gmpy implementation is preferred because it is significantly faster.

class sympy.polys.domains.**RationalField**

Abstract base class for the domain *QQ* (page 2529).

The *RationalField* (page 2530) class represents the field of rational numbers $\mathbb{Q}$ as a *Domain* (page 2504) in the domain system. *RationalField* (page 2530) is a superclass of *PythonRationalField* (page 2531) and *GMPYRationalField* (page 2531) one of which will be the implementation for *QQ* (page 2529) depending on whether either of gmpy or gmpy2 is installed or not.

See also:

Domain (page 2504)

algebraic_field(**extension, alias=None*)

Returns an algebraic field, i.e. $\mathbb{Q}(\alpha, \ldots)$.

Parameters

***extension** : One or more *Expr* (page 947)

Generators of the extension. These should be expressions that are algebraic over $\mathbb{Q}$.

alias : str, *Symbol* (page 976), None, optional (default=None)

If provided, this will be used as the alias symbol for the primitive element of the returned *AlgebraicField* (page 2539).

Returns

AlgebraicField (page 2539)

A *Domain* (page 2504) representing the algebraic field extension.

Examples

```
>>> from sympy import QQ, sqrt
>>> QQ.algebraic_field(sqrt(2))
QQ<sqrt(2)>
```

denom(*a*)

Returns denominator of a.

div(*a, b*)

Division of a and b, implies __truediv__.

exquo(*a*, *b*)
Exact quotient of a and b, implies __truediv__.

from_AlgebraicField(*a*, *K0*)
Convert a *ANP* (page 2568) object to *QQ* (page 2529).

See *convert()* (page 2508)

from_GaussianRationalField(*a*, *K0*)
Convert a GaussianElement object to dtype.

from_QQ(*a*, *K0*)
Convert a Python Fraction object to dtype.

from_QQ_gmpy(*a*, *K0*)
Convert a GMPY mpq object to dtype.

from_QQ_python(*a*, *K0*)
Convert a Python Fraction object to dtype.

from_RealField(*a*, *K0*)
Convert a mpmath mpf object to dtype.

from_ZZ(*a*, *K0*)
Convert a Python int object to dtype.

from_ZZ_gmpy(*a*, *K0*)
Convert a GMPY mpz object to dtype.

from_ZZ_python(*a*, *K0*)
Convert a Python int object to dtype.

from_sympy(*a*)
Convert SymPy's Integer to dtype.

get_ring()
Returns ring associated with self.

numer(*a*)
Returns numerator of a.

quo(*a*, *b*)
Quotient of a and b, implies __truediv__.

rem(*a*, *b*)
Remainder of a and b, implies nothing.

to_sympy(*a*)
Convert a to a SymPy object.

class sympy.polys.domains.**PythonRationalField**
Rational field based on *MPQ* (page 2533).

This will be used as *QQ* (page 2529) if gmpy and gmpy2 are not installed. Elements are instances of *MPQ* (page 2533).

class sympy.polys.domains.**GMPYRationalField**

Rational field based on GMPY's mpq type.

This will be the implementation of *QQ* (page 2529) if gmpy or gmpy2 is installed. Elements will be of type gmpy.mpq.

denom(*a*)
Returns denominator of a.

div(*a*, *b*)
Division of a and b, implies __truediv__.

exquo(*a*, *b*)
Exact quotient of a and b, implies __truediv__.

factorial(*a*)
Returns factorial of a.

from_GaussianRationalField(*a*, *K0*)
Convert a GaussianElement object to dtype.

from_QQ_gmpy(*a*, *K0*)
Convert a GMPY mpq object to dtype.

from_QQ_python(*a*, *K0*)
Convert a Python Fraction object to dtype.

from_RealField(*a*, *K0*)
Convert a mpmath mpf object to dtype.

from_ZZ_gmpy(*a*, *K0*)
Convert a GMPY mpz object to dtype.

from_ZZ_python(*a*, *K0*)
Convert a Python int object to dtype.

from_sympy(*a*)
Convert SymPy's Integer to dtype.

get_ring()
Returns ring associated with self.

numer(*a*)
Returns numerator of a.

quo(*a*, *b*)
Quotient of a and b, implies __truediv__.

rem(*a*, *b*)
Remainder of a and b, implies nothing.

to_sympy(*a*)
Convert a to a SymPy object.

class sympy.external.pythonmpq.**PythonMPQ**(*numerator*, *denominator=None*)

Rational number implementation that is intended to be compatible with gmpy2's mpq.

Also slightly faster than fractions.Fraction.

PythonMPQ should be treated as immutable although no effort is made to prevent mutation (since that might slow down calculations).

MPQ

The MPQ type is either *PythonMPQ* (page 2532) or otherwise the mpq type from gmpy2.

Gaussian domains

The Gaussian domains *ZZ_I* (page 2534) and *QQ_I* (page 2536) share common superclasses *GaussianElement* (page 2533) for the domain elements and *GaussianDomain* (page 2533) for the domains themselves.

class sympy.polys.domains.gaussiandomains.**GaussianDomain**

Base class for Gaussian domains.

from_AlgebraicField(*a*, *K0*)

Convert an element from ZZ<I> or QQ<I> to self.dtype.

from_QQ(*a*, *K0*)

Convert a GMPY mpq to self.dtype.

from_QQ_gmpy(*a*, *K0*)

Convert a GMPY mpq to self.dtype.

from_QQ_python(*a*, *K0*)

Convert a QQ_python element to self.dtype.

from_ZZ(*a*, *K0*)

Convert a ZZ_python element to self.dtype.

from_ZZ_gmpy(*a*, *K0*)

Convert a GMPY mpz to self.dtype.

from_ZZ_python(*a*, *K0*)

Convert a ZZ_python element to self.dtype.

from_sympy(*a*)

Convert a SymPy object to self.dtype.

inject(**gens*)

Inject generators into this domain.

is_negative(*element*)

Returns False for any GaussianElement.

is_nonnegative(*element*)

Returns False for any GaussianElement.

is_nonpositive(*element*)

Returns False for any GaussianElement.

is_positive(*element*)

Returns False for any GaussianElement.

to_sympy(*a*)

Convert a to a SymPy object.

class sympy.polys.domains.gaussiandomains.**GaussianElement**(*x*, *y=0*)

Base class for elements of Gaussian type domains.

classmethod new(*x*, *y*)

Create a new GaussianElement of the same domain.

parent()

The domain that this is an element of (ZZ_I or QQ_I)

quadrant()

Return quadrant index 0-3.

0 is included in quadrant 0.

ZZ_I

class sympy.polys.domains.gaussiandomains.**GaussianIntegerRing**

Ring of Gaussian integers ZZ_I

The *ZZ_I* (page 2534) domain represents the Gaussian integers $\mathbb{Z}[i]$ as a *Domain* (page 2504) in the domain system (see *Introducing the Domains of the poly module* (page 2477)).

By default a *Poly* (page 2378) created from an expression with coefficients that are combinations of integers and I ($\sqrt{-1}$) will have the domain *ZZ_I* (page 2534).

```
>>> from sympy import Poly, Symbol, I
>>> x = Symbol('x')
>>> p = Poly(x**2 + I)
>>> p
Poly(x**2 + I, x, domain='ZZ_I')
>>> p.domain
ZZ_I
```

The *ZZ_I* (page 2534) domain can be used to factorise polynomials that are reducible over the Gaussian integers.

```
>>> from sympy import factor
>>> factor(x**2 + 1)
x**2 + 1
>>> factor(x**2 + 1, domain='ZZ_I')
(x - I)*(x + I)
```

The corresponding field of fractions is the domain of the Gaussian rationals *QQ_I* (page 2536). Conversely *ZZ_I* (page 2534) is the ring of integers of *QQ_I* (page 2536).

```
>>> from sympy import ZZ_I, QQ_I
>>> ZZ_I.get_field()
QQ_I
>>> QQ_I.get_ring()
ZZ_I
```

When using the domain directly *ZZ_I* (page 2534) can be used as a constructor.

```
>>> ZZ_I(3, 4)
(3 + 4*I)
>>> ZZ_I(5)
(5 + 0*I)
```

The domain elements of *ZZ_I* (page 2534) are instances of `GaussianInteger` (page 2536) which support the rings operations `+`, `-`, `*`, `**`.

```
>>> z1 = ZZ_I(5, 1)
>>> z2 = ZZ_I(2, 3)
>>> z1
(5 + 1*I)
>>> z2
(2 + 3*I)
>>> z1 + z2
(7 + 4*I)
>>> z1 * z2
(7 + 17*I)
>>> z1 ** 2
(24 + 10*I)
```

Both floor (`//`) and modulo (`%`) division work with `GaussianInteger` (page 2536) (see the `div()` (page 2509) method).

```
>>> z3, z4 = ZZ_I(5), ZZ_I(1, 3)
>>> z3 // z4  # floor division
(1 + -1*I)
>>> z3 % z4   # modulo division (remainder)
(1 + -2*I)
>>> (z3//z4)*z4 + z3%z4 == z3
True
```


True division (`/`) in *ZZ_I* (page 2534) gives an element of *QQ_I* (page 2536). The `exquo()` (page 2511) method can be used to divide in *ZZ_I* (page 2534) when exact division is possible.

```
>>> z1 / z2
(1 + -1*I)
>>> ZZ_I.exquo(z1, z2)
(1 + -1*I)
>>> z3 / z4
(1/2 + -3/2*I)
>>> ZZ_I.exquo(z3, z4)
Traceback (most recent call last):
    ...
ExactQuotientFailed: (1 + 3*I) does not divide (5 + 0*I) in ZZ_I
```

The `gcd()` (page 2514) method can be used to compute the gcd of any two elements.

```
>>> ZZ_I.gcd(ZZ_I(10), ZZ_I(2))
(2 + 0*I)
>>> ZZ_I.gcd(ZZ_I(5), ZZ_I(2, 1))
(2 + 1*I)
```

dtype
alias of *GaussianInteger* (page 2536)

from_GaussianIntegerRing(*a*, *K0*)
Convert a ZZ_I element to ZZ_I.

from_GaussianRationalField(*a*, *K0*)
Convert a QQ_I element to ZZ_I.

gcd(*a*, *b*)
Greatest common divisor of a and b over ZZ_I.

get_field()
Returns a field associated with `self`.

get_ring()
Returns a ring associated with `self`.

lcm(*a*, *b*)
Least common multiple of a and b over ZZ_I.

normalize(*d*, **args*)
Return first quadrant element associated with `d`.

Also multiply the other arguments by the same power of i.

class sympy.polys.domains.gaussiandomains.**GaussianInteger**(*x*, *y=0*)
Gaussian integer: domain element for *ZZ_I* (page 2534)

```
>>> from sympy import ZZ_I
>>> z = ZZ_I(2, 3)
>>> z
(2 + 3*I)
>>> type(z)
<class 'sympy.polys.domains.gaussiandomains.GaussianInteger'>
```

QQ_I

class sympy.polys.domains.gaussiandomains.**GaussianRationalField**
Field of Gaussian rationals `QQ_I`

The *QQ_I* (page 2536) domain represents the Gaussian rationals $\mathbb{Q}(i)$ as a *Domain* (page 2504) in the domain system (see *Introducing the Domains of the poly module* (page 2477)).

By default a *Poly* (page 2378) created from an expression with coefficients that are combinations of rationals and `I` ($\sqrt{-1}$) will have the domain *QQ_I* (page 2536).

```
>>> from sympy import Poly, Symbol, I
>>> x = Symbol('x')
>>> p = Poly(x**2 + I/2)
>>> p
Poly(x**2 + I/2, x, domain='QQ_I')
>>> p.domain
QQ_I
```

The polys option `gaussian=True` can be used to specify that the domain should be *QQ_I* (page 2536) even if the coefficients do not contain `I` or are all integers.

```
>>> Poly(x**2)
Poly(x**2, x, domain='ZZ')
>>> Poly(x**2 + I)
Poly(x**2 + I, x, domain='ZZ_I')
>>> Poly(x**2/2)
Poly(1/2*x**2, x, domain='QQ')
>>> Poly(x**2, gaussian=True)
Poly(x**2, x, domain='QQ_I')
>>> Poly(x**2 + I, gaussian=True)
Poly(x**2 + I, x, domain='QQ_I')
>>> Poly(x**2/2, gaussian=True)
Poly(1/2*x**2, x, domain='QQ_I')
```

The *QQ_I* (page 2536) domain can be used to factorise polynomials that are reducible over the Gaussian rationals.

```
>>> from sympy import factor, QQ_I
>>> factor(x**2/4 + 1)
(x**2 + 4)/4
>>> factor(x**2/4 + 1, domain='QQ_I')
(x - 2*I)*(x + 2*I)/4
>>> factor(x**2/4 + 1, domain=QQ_I)
(x - 2*I)*(x + 2*I)/4
```

It is also possible to specify the *QQ_I* (page 2536) domain explicitly with polys functions like `apart()` (page 2443).

```
>>> from sympy import apart
>>> apart(1/(1 + x**2))
1/(x**2 + 1)
>>> apart(1/(1 + x**2), domain=QQ_I)
I/(2*(x + I)) - I/(2*(x - I))
```

The corresponding ring of integers is the domain of the Gaussian integers *ZZ_I* (page 2534). Conversely *QQ_I* (page 2536) is the field of fractions of *ZZ_I* (page 2534).

```
>>> from sympy import ZZ_I, QQ_I, QQ
>>> ZZ_I.get_field()
QQ_I
>>> QQ_I.get_ring()
ZZ_I
```

When using the domain directly *QQ_I* (page 2536) can be used as a constructor.

```
>>> QQ_I(3, 4)
(3 + 4*I)
>>> QQ_I(5)
(5 + 0*I)
>>> QQ_I(QQ(2, 3), QQ(4, 5))
(2/3 + 4/5*I)
```

The domain elements of *QQ_I* (page 2536) are instances of `GaussianRational` (page 2539) which support the field operations `+,-,*,**,/`.

```
>>> z1 = QQ_I(5, 1)
>>> z2 = QQ_I(2, QQ(1, 2))
>>> z1
(5 + 1*I)
>>> z2
(2 + 1/2*I)
>>> z1 + z2
(7 + 3/2*I)
>>> z1 * z2
(19/2 + 9/2*I)
>>> z2 ** 2
(15/4 + 2*I)
```

True division (`/`) in *QQ_I* (page 2536) gives an element of *QQ_I* (page 2536) and is always exact.

```
>>> z1 / z2
(42/17 + -2/17*I)
>>> QQ_I.exquo(z1, z2)
(42/17 + -2/17*I)
>>> z1 == (z1/z2)*z2
True
```

Both floor (`//`) and modulo (`%`) division can be used with `GaussianRational` (page 2539) (see `div()` (page 2509)) but division is always exact so there is no remainder.


```
>>> z1 // z2
(42/17 + -2/17*I)
>>> z1 % z2
(0 + 0*I)
>>> QQ_I.div(z1, z2)
((42/17 + -2/17*I), (0 + 0*I))
>>> (z1//z2)*z2 + z1%z2 == z1
True
```

as_AlgebraicField()

Get equivalent domain as an `AlgebraicField`.

denom(*a*)

Get the denominator of `a`.

dtype

alias of `GaussianRational` (page 2539)

from_GaussianIntegerRing(*a, K0*)

Convert a ZZ_I element to QQ_I.

from_GaussianRationalField(*a, K0*)

Convert a QQ_I element to QQ_I.

get_field()

Returns a field associated with `self`.

get_ring()

Returns a ring associated with self.

numer(*a*)

Get the numerator of a.

class sympy.polys.domains.gaussiandomains.**GaussianRational**(*x*, *y=0*)

Gaussian rational: domain element for *QQ_I* (page 2536)

```
>>> from sympy import QQ_I, QQ
>>> z = QQ_I(QQ(2, 3), QQ(4, 5))
>>> z
(2/3 + 4/5*I)
>>> type(z)
<class 'sympy.polys.domains.gaussiandomains.GaussianRational'>
```

QQ<a>

class sympy.polys.domains.**AlgebraicField**(*dom*, **ext*, *alias=None*)

Algebraic number field *QQ<a>* (page 2539)

A *QQ<a>* (page 2539) domain represents an algebraic number field $\mathbb{Q}(a)$ as a *Domain* (page 2504) in the domain system (see *Introducing the Domains of the poly module* (page 2477)).

A *Poly* (page 2378) created from an expression involving algebraic numbers will treat the algebraic numbers as generators if the generators argument is not specified.

```
>>> from sympy import Poly, Symbol, sqrt
>>> x = Symbol('x')
>>> Poly(x**2 + sqrt(2))
Poly(x**2 + (sqrt(2)), x, sqrt(2), domain='ZZ')
```

That is a multivariate polynomial with sqrt(2) treated as one of the generators (variables). If the generators are explicitly specified then sqrt(2) will be considered to be a coefficient but by default the *EX* (page 2549) domain is used. To make a *Poly* (page 2378) with a *QQ<a>* (page 2539) domain the argument extension=True can be given.

```
>>> Poly(x**2 + sqrt(2), x)
Poly(x**2 + sqrt(2), x, domain='EX')
>>> Poly(x**2 + sqrt(2), x, extension=True)
Poly(x**2 + sqrt(2), x, domain='QQ<sqrt(2)>')
```

A generator of the algebraic field extension can also be specified explicitly which is particularly useful if the coefficients are all rational but an extension field is needed (e.g. to factor the polynomial).

```
>>> Poly(x**2 + 1)
Poly(x**2 + 1, x, domain='ZZ')
>>> Poly(x**2 + 1, extension=sqrt(2))
Poly(x**2 + 1, x, domain='QQ<sqrt(2)>')
```

It is possible to factorise a polynomial over a *QQ<a>* (page 2539) domain using the extension argument to *factor()* (page 2373) or by specifying the domain explicitly.

```
>>> from sympy import factor, QQ
>>> factor(x**2 - 2)
x**2 - 2
>>> factor(x**2 - 2, extension=sqrt(2))
(x - sqrt(2))*(x + sqrt(2))
>>> factor(x**2 - 2, domain='QQ<sqrt(2)>')
(x - sqrt(2))*(x + sqrt(2))
>>> factor(x**2 - 2, domain=QQ.algebraic_field(sqrt(2)))
(x - sqrt(2))*(x + sqrt(2))
```

The `extension=True` argument can be used but will only create an extension that contains the coefficients which is usually not enough to factorise the polynomial.

```
>>> p = x**3 + sqrt(2)*x**2 - 2*x - 2*sqrt(2)
>>> factor(p)                         # treats sqrt(2) as a symbol
(x + sqrt(2))*(x**2 - 2)
>>> factor(p, extension=True)
(x - sqrt(2))*(x + sqrt(2))**2
>>> factor(x**2 - 2, extension=True)  # all rational coefficients
x**2 - 2
```

It is also possible to use *QQ<a>* (page 2539) with the `cancel()` (page 2376) and `gcd()` (page 2368) functions.

```
>>> from sympy import cancel, gcd
>>> cancel((x**2 - 2)/(x - sqrt(2)))
(x**2 - 2)/(x - sqrt(2))
>>> cancel((x**2 - 2)/(x - sqrt(2)), extension=sqrt(2))
x + sqrt(2)
>>> gcd(x**2 - 2, x - sqrt(2))
1
>>> gcd(x**2 - 2, x - sqrt(2), extension=sqrt(2))
x - sqrt(2)
```

When using the domain directly *QQ<a>* (page 2539) can be used as a constructor to create instances which then support the operations `+,-,*,**,/`. The `algebraic_field()` (page 2508) method is used to construct a particular *QQ<a>* (page 2539) domain. The `from_sympy()` (page 2513) method can be used to create domain elements from normal SymPy expressions.

```
>>> K = QQ.algebraic_field(sqrt(2))
>>> K
QQ<sqrt(2)>
>>> xk = K.from_sympy(3 + 4*sqrt(2))
>>> xk
ANP([4, 3], [1, 0, -2], QQ)
```

Elements of *QQ<a>* (page 2539) are instances of `ANP` (page 2568) which have limited printing support. The raw display shows the internal representation of the element as the list `[4, 3]` representing the coefficients of `1` and `sqrt(2)` for this element in the form `a * sqrt(2) + b * 1` where `a` and `b` are elements of *QQ* (page 2529). The minimal polynomial for the generator `(x**2 - 2)` is also shown in the *DUP representation* (page 2479) as the list `[1, 0, -2]`. We can use `to_sympy()` (page 2517) to get a better printed form for the elements and to see the results of operations.

```
>>> xk = K.from_sympy(3 + 4*sqrt(2))
>>> yk = K.from_sympy(2 + 3*sqrt(2))
>>> xk * yk
ANP([17, 30], [1, 0, -2], QQ)
>>> K.to_sympy(xk * yk)
17*sqrt(2) + 30
>>> K.to_sympy(xk + yk)
5 + 7*sqrt(2)
>>> K.to_sympy(xk ** 2)
24*sqrt(2) + 41
>>> K.to_sympy(xk / yk)
sqrt(2)/14 + 9/7
```

Any expression representing an algebraic number can be used to generate a *QQ<a>* (page 2539) domain provided its minimal polynomial can be computed. The function *minpoly()* (page 2711) function is used for this.

```
>>> from sympy import exp, I, pi, minpoly
>>> g = exp(2*I*pi/3)
>>> g
exp(2*I*pi/3)
>>> g.is_algebraic
True
>>> minpoly(g, x)
x**2 + x + 1
>>> factor(x**3 - 1, extension=g)
(x - 1)*(x - exp(2*I*pi/3))*(x + 1 + exp(2*I*pi/3))
```

It is also possible to make an algebraic field from multiple extension elements.

```
>>> K = QQ.algebraic_field(sqrt(2), sqrt(3))
>>> K
QQ<sqrt(2) + sqrt(3)>
>>> p = x**4 - 5*x**2 + 6
>>> factor(p)
(x**2 - 3)*(x**2 - 2)
>>> factor(p, domain=K)
(x - sqrt(2))*(x + sqrt(2))*(x - sqrt(3))*(x + sqrt(3))
>>> factor(p, extension=[sqrt(2), sqrt(3)])
(x - sqrt(2))*(x + sqrt(2))*(x - sqrt(3))*(x + sqrt(3))
```

Multiple extension elements are always combined together to make a single primitive element. In the case of [sqrt(2), sqrt(3)] the primitive element chosen is sqrt(2) + sqrt(3) which is why the domain displays as QQ<sqrt(2) + sqrt(3)>. The minimal polynomial for the primitive element is computed using the *primitive_element()* (page 2712) function.

```
>>> from sympy import primitive_element
>>> primitive_element([sqrt(2), sqrt(3)], x)
(x**4 - 10*x**2 + 1, [1, 1])
>>> minpoly(sqrt(2) + sqrt(3), x)
x**4 - 10*x**2 + 1
```

The extension elements that generate the domain can be accessed from the domain using the *ext* (page 2543) and *orig_ext* (page 2545) attributes as instances of *AlgebraicNumber* (page 988). The minimal polynomial for the primitive element as a *DMP* (page 2561) instance is available as *mod* (page 2545).

```
>>> K = QQ.algebraic_field(sqrt(2), sqrt(3))
>>> K
QQ<sqrt(2) + sqrt(3)>
>>> K.ext
sqrt(2) + sqrt(3)
>>> K.orig_ext
(sqrt(2), sqrt(3))
>>> K.mod
DMP([1, 0, -10, 0, 1], QQ, None)
```

The discriminant of the field can be obtained from the *discriminant()* (page 2543) method, and an integral basis from the *integral_basis()* (page 2544) method. The latter returns a list of *ANP* (page 2568) instances by default, but can be made to return instances of *Expr* (page 947) or *AlgebraicNumber* (page 988) by passing a `fmt` argument. The maximal order, or ring of integers, of the field can also be obtained from the *maximal_order()* (page 2545) method, as a *Submodule* (page 2726).

```
>>> zeta5 = exp(2*I*pi/5)
>>> K = QQ.algebraic_field(zeta5)
>>> K
QQ<exp(2*I*pi/5)>
>>> K.discriminant()
125
>>> K = QQ.algebraic_field(sqrt(5))
>>> K
QQ<sqrt(5)>
>>> K.integral_basis(fmt='sympy')
[1, 1/2 + sqrt(5)/2]
>>> K.maximal_order()
Submodule[[2, 0], [1, 1]]/2
```

The factorization of a rational prime into prime ideals of the field is computed by the *primes_above()* (page 2545) method, which returns a list of *PrimeIdeal* (page 2706) instances.

```
>>> zeta7 = exp(2*I*pi/7)
>>> K = QQ.algebraic_field(zeta7)
>>> K
QQ<exp(2*I*pi/7)>
>>> K.primes_above(11)
[(11, _x**3 + 5*_x**2 + 4*_x - 1), (11, _x**3 - 4*_x**2 - 5*_x - 1)]
```

Notes

It is not currently possible to generate an algebraic extension over any domain other than *QQ* (page 2529). Ideally it would be possible to generate extensions like `QQ(x)(sqrt(x**2 - 2))`. This is equivalent to the quotient ring `QQ(x)[y]/(y**2 - x**2 + 2)` and there are two implementations of this kind of quotient ring/extension in the *QuotientRing* (page 2551) and *MonogenicFiniteExtension* (page 2475) classes. Each of those implementations needs some work to make them fully usable though.

algebraic_field(**extension, alias=None*)

Returns an algebraic field, i.e. $\mathbb{Q}(\alpha, \ldots)$.

denom(*a*)

Returns denominator of `a`.

discriminant()

Get the discriminant of the field.

dtype

alias of *ANP* (page 2568)

ext

Primitive element used for the extension.

```
>>> from sympy import QQ, sqrt
>>> K = QQ.algebraic_field(sqrt(2), sqrt(3))
>>> K.ext
sqrt(2) + sqrt(3)
```


from_AlgebraicField(*a, K0*)

Convert AlgebraicField element 'a' to another AlgebraicField

from_GaussianIntegerRing(*a, K0*)

Convert a GaussianInteger element 'a' to `dtype`.

from_GaussianRationalField(*a, K0*)

Convert a GaussianRational element 'a' to `dtype`.

from_QQ(*a, K0*)

Convert a Python `Fraction` object to `dtype`.

from_QQ_gmpy(*a, K0*)

Convert a GMPY `mpq` object to `dtype`.

from_QQ_python(*a, K0*)

Convert a Python `Fraction` object to `dtype`.

from_RealField(*a, K0*)

Convert a mpmath `mpf` object to `dtype`.

from_ZZ(*a, K0*)

Convert a Python `int` object to `dtype`.

from_ZZ_gmpy(*a, K0*)

Convert a GMPY `mpz` object to `dtype`.

from_ZZ_python(*a*, *K0*)
Convert a Python int object to dtype.

from_sympy(*a*)
Convert SymPy's expression to dtype.

get_ring()
Returns a ring associated with self.

integral_basis(*fmt=None*)
Get an integral basis for the field.

Parameters
fmt : str, None, optional (default=None)

If None, return a list of *ANP* (page 2568) instances. If "sympy", convert each element of the list to an *Expr* (page 947), using self.to_sympy(). If "alg", convert each element of the list to an *AlgebraicNumber* (page 988), using self.to_alg_num().

Examples

```
>>> from sympy import QQ, AlgebraicNumber, sqrt
>>> alpha = AlgebraicNumber(sqrt(5), alias='alpha')
>>> k = QQ.algebraic_field(alpha)
>>> B0 = k.integral_basis()
>>> B1 = k.integral_basis(fmt='sympy')
>>> B2 = k.integral_basis(fmt='alg')
>>> print(B0[1])
ANP([mpq(1,2), mpq(1,2)], [mpq(1,1), mpq(0,1), mpq(-5,1)], QQ)
>>> print(B1[1])
1/2 + alpha/2
>>> print(B2[1])
alpha/2 + 1/2
```

In the last two cases we get legible expressions, which print somewhat differently because of the different types involved:

```
>>> print(type(B1[1]))
<class 'sympy.core.add.Add'>
>>> print(type(B2[1]))
<class 'sympy.core.numbers.AlgebraicNumber'>
```

See also:

to_sympy (page 2545), *to_alg_num* (page 2545), *maximal_order* (page 2545)

is_negative(*a*)
Returns True if a is negative.

is_nonnegative(*a*)
Returns True if a is non-negative.

is_nonpositive(*a*)
Returns True if a is non-positive.

is_positive(*a*)

Returns True if a is positive.

maximal_order()

Compute the maximal order, or ring of integers, of the field.

Returns

Submodule (page 2726).

See also:

integral_basis (page 2544)

mod

Minimal polynomial for the primitive element of the extension.

```
>>> from sympy import QQ, sqrt
>>> K = QQ.algebraic_field(sqrt(2))
>>> K.mod
DMP([1, 0, -2], QQ, None)
```

numer(*a*)

Returns numerator of a.

orig_ext

Original elements given to generate the extension.

```
>>> from sympy import QQ, sqrt
>>> K = QQ.algebraic_field(sqrt(2), sqrt(3))
>>> K.orig_ext
(sqrt(2), sqrt(3))
```

primes_above(*p*)

Compute the prime ideals lying above a given rational prime *p*.

to_alg_num(*a*)

Convert a of dtype to an *AlgebraicNumber* (page 988).

to_sympy(*a*)

Convert a of dtype to a SymPy object.

RR

class sympy.polys.domains.**RealField**(*prec=53*, *dps=None*, *tol=None*)

Real numbers up to the given precision.

almosteq(*a*, *b*, *tolerance=None*)

Check if a and b are almost equal.

from_sympy(*expr*)

Convert SymPy's number to dtype.

gcd(*a*, *b*)

Returns GCD of a and b.

get_exact()
Returns an exact domain associated with `self`.

get_ring()
Returns a ring associated with `self`.

lcm(*a*, *b*)
Returns LCM of `a` and `b`.

to_rational(*element*, *limit=True*)
Convert a real number to rational number.

to_sympy(*element*)
Convert `element` to SymPy number.

class sympy.polys.domains.mpelements.**RealElement**(*val=(0, 0, 0, 0)*, ***kwargs*)
An element of a real domain.

CC

class sympy.polys.domains.**ComplexField**(*prec=53*, *dps=None*, *tol=None*)
Complex numbers up to the given precision.

almosteq(*a*, *b*, *tolerance=None*)
Check if `a` and `b` are almost equal.

from_sympy(*expr*)
Convert SymPy's number to `dtype`.

gcd(*a*, *b*)
Returns GCD of `a` and `b`.

get_exact()
Returns an exact domain associated with `self`.

get_ring()
Returns a ring associated with `self`.

is_negative(*element*)
Returns `False` for any `ComplexElement`.

is_nonnegative(*element*)
Returns `False` for any `ComplexElement`.

is_nonpositive(*element*)
Returns `False` for any `ComplexElement`.

is_positive(*element*)
Returns `False` for any `ComplexElement`.

lcm(*a*, *b*)
Returns LCM of `a` and `b`.

to_sympy(*element*)
Convert `element` to SymPy number.

class sympy.polys.domains.mpelements.**ComplexElement**(*real=0*, *imag=0*)
An element of a complex domain.

K[x]

class sympy.polys.domains.**PolynomialRing**(*domain_or_ring*, *symbols=None*, *order=None*)

A class for representing multivariate polynomial rings.

factorial(*a*)

Returns factorial of a.

from_AlgebraicField(*a*, *K0*)

Convert an algebraic number to `dtype`.

from_ComplexField(*a*, *K0*)

Convert a mpmath mpf object to $dtype$.

from_FractionField(*a*, *K0*)

Convert a rational function to `dtype`.

from_GaussianIntegerRing(*a*, *K0*)

Convert a $GaussianInteger$ object to $dtype$.

from_GaussianRationalField(*a*, *K0*)

Convert a $GaussianRational$ object to $dtype$.

from_GlobalPolynomialRing(*a*, *K0*)

Convert from old poly ring to `dtype`.

from_PolynomialRing(*a*, *K0*)

Convert a polynomial to `dtype`.

from_QQ(*a*, *K0*)

Convert a Python $Fraction$ object to $dtype$.

from_QQ_gmpy(*a*, *K0*)

Convert a GMPY mpq object to $dtype$.

from_QQ_python(*a*, *K0*)

Convert a Python $Fraction$ object to $dtype$.

from_RealField(*a*, *K0*)

Convert a mpmath mpf object to $dtype$.

from_ZZ(*a*, *K0*)

Convert a Python int object to $dtype$.

from_ZZ_gmpy(*a*, *K0*)

Convert a GMPY mpz object to $dtype$.

from_ZZ_python(*a*, *K0*)

Convert a Python int object to $dtype$.

from_sympy(*a*)

Convert SymPy's expression to $dtype$.

gcd(*a*, *b*)

Returns GCD of a and b.

gcdex(*a*, *b*)
Extended GCD of a and b.

get_field()
Returns a field associated with $self$.

is_negative(*a*)
Returns True if $LC(a)$ is negative.

is_nonnegative(*a*)
Returns True if $LC(a)$ is non-negative.

is_nonpositive(*a*)
Returns True if $LC(a)$ is non-positive.

is_positive(*a*)
Returns True if $LC(a)$ is positive.

is_unit(*a*)
Returns `True` if `a` is a unit of `self`

lcm(*a*, *b*)
Returns LCM of a and b.

to_sympy(*a*)
Convert a to a SymPy object.

K(x)

class `sympy.polys.domains.`**FractionField**(*domain_or_field*, *symbols=None*, *order=None*)
A class for representing multivariate rational function fields.

denom(*a*)
Returns denominator of `a`.

factorial(*a*)
Returns factorial of `a`.

from_AlgebraicField(*a*, *K0*)
Convert an algebraic number to `dtype`.

from_ComplexField(*a*, *K0*)
Convert a mpmath `mpf` object to `dtype`.

from_FractionField(*a*, *K0*)
Convert a rational function to `dtype`.

from_GaussianIntegerRing(*a*, *K0*)
Convert a `GaussianInteger` object to `dtype`.

from_GaussianRationalField(*a*, *K0*)
Convert a `GaussianRational` object to `dtype`.

from_PolynomialRing(*a*, *K0*)
Convert a polynomial to `dtype`.

from_QQ(*a*, *K0*)
Convert a Python Fraction object to dtype.

from_QQ_gmpy(*a*, *K0*)
Convert a GMPY mpq object to dtype.

from_QQ_python(*a*, *K0*)
Convert a Python Fraction object to dtype.

from_RealField(*a*, *K0*)
Convert a mpmath mpf object to dtype.

from_ZZ(*a*, *K0*)
Convert a Python int object to dtype.

from_ZZ_gmpy(*a*, *K0*)
Convert a GMPY mpz object to dtype.

from_ZZ_python(*a*, *K0*)
Convert a Python int object to dtype.

from_sympy(*a*)
Convert SymPy's expression to dtype.

get_ring()
Returns a field associated with self.

is_negative(*a*)
Returns True if LC(a) is negative.

is_nonnegative(*a*)
Returns True if LC(a) is non-negative.

is_nonpositive(*a*)
Returns True if LC(a) is non-positive.

is_positive(*a*)
Returns True if LC(a) is positive.

numer(*a*)
Returns numerator of a.

to_sympy(*a*)
Convert a to a SymPy object.

EX

class sympy.polys.domains.**ExpressionDomain**
A class for arbitrary expressions.

class Expression(*ex*)
An arbitrary expression.

denom(*a*)
Returns denominator of a.

dtype
alias of *Expression* (page 2551)

from_ExpressionDomain(*a*, *K0*)
Convert a EX object to dtype.

from_FractionField(*a*, *K0*)
Convert a DMF object to dtype.

from_GaussianIntegerRing(*a*, *K0*)
Convert a GaussianRational object to dtype.

from_GaussianRationalField(*a*, *K0*)
Convert a GaussianRational object to dtype.

from_PolynomialRing(*a*, *K0*)
Convert a DMP object to dtype.

from_QQ(*a*, *K0*)
Convert a Python Fraction object to dtype.

from_QQ_gmpy(*a*, *K0*)
Convert a GMPY mpq object to dtype.

from_QQ_python(*a*, *K0*)
Convert a Python Fraction object to dtype.

from_RealField(*a*, *K0*)
Convert a mpmath mpf object to dtype.

from_ZZ(*a*, *K0*)
Convert a Python int object to dtype.

from_ZZ_gmpy(*a*, *K0*)
Convert a GMPY mpz object to dtype.

from_ZZ_python(*a*, *K0*)
Convert a Python int object to dtype.

from_sympy(*a*)
Convert SymPy's expression to dtype.

get_field()
Returns a field associated with self.

get_ring()
Returns a ring associated with self.

is_negative(*a*)
Returns True if a is negative.

is_nonnegative(*a*)
Returns True if a is non-negative.

is_nonpositive(*a*)
Returns True if a is non-positive.

is_positive(*a*)

Returns True if a is positive.

numer(*a*)

Returns numerator of a.

to_sympy(*a*)

Convert a to a SymPy object.

class ExpressionDomain.**Expression**(*ex*)

An arbitrary expression.

Quotient ring

class sympy.polys.domains.quotientring.**QuotientRing**(*ring*, *ideal*)

Class representing (commutative) quotient rings.

You should not usually instantiate this by hand, instead use the constructor from the base ring in the construction.

```
>>> from sympy.abc import x
>>> from sympy import QQ
>>> I = QQ.old_poly_ring(x).ideal(x**3 + 1)
>>> QQ.old_poly_ring(x).quotient_ring(I)
QQ[x]/<x**3 + 1>
```

Shorter versions are possible:

```
>>> QQ.old_poly_ring(x)/I
QQ[x]/<x**3 + 1>
```

```
>>> QQ.old_poly_ring(x)/[x**3 + 1]
QQ[x]/<x**3 + 1>
```

Attributes:

- ring - the base ring
- base_ideal - the ideal used to form the quotient

Sparse polynomials

Sparse polynomials are represented as dictionaries.

sympy.polys.rings.**ring**(*symbols*, *domain*, *order=LexOrder()*)

Construct a polynomial ring returning `(ring, x_1, ..., x_n)`.

Parameters

symbols : str

Symbol/Expr or sequence of str, Symbol/Expr (non-empty)

domain : *Domain* (page 2504) or coercible

order : *MonomialOrder* (page 2431) or coercible, optional, defaults to `lex`

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.orderings import lex
```

```
>>> R, x, y, z = ring("x,y,z", ZZ, lex)
>>> R
Polynomial ring in x, y, z over ZZ with lex order
>>> x + y + z
x + y + z
>>> type(_)
<class 'sympy.polys.rings.PolyElement'>
```

sympy.polys.rings.**xring**(*symbols*, *domain*, *order=LexOrder()*)

Construct a polynomial ring returning `(ring, (x_1, ..., x_n))`.

Parameters

symbols : str

Symbol/Expr or sequence of str, Symbol/Expr (non-empty)

domain : *Domain* (page 2504) or coercible

order : *MonomialOrder* (page 2431) or coercible, optional, defaults to `lex`

Examples

```
>>> from sympy.polys.rings import xring
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.orderings import lex
```

```
>>> R, (x, y, z) = xring("x,y,z", ZZ, lex)
>>> R
Polynomial ring in x, y, z over ZZ with lex order
>>> x + y + z
x + y + z
>>> type(_)
<class 'sympy.polys.rings.PolyElement'>
```

sympy.polys.rings.**vring**(*symbols*, *domain*, *order=LexOrder()*)

Construct a polynomial ring and inject `x_1, ..., x_n` into the global namespace.

Parameters

symbols : str

Symbol/Expr or sequence of str, Symbol/Expr (non-empty)

domain : *Domain* (page 2504) or coercible

order : *MonomialOrder* (page 2431) or coercible, optional, defaults to `lex`

Examples

```
>>> from sympy.polys.rings import vring
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.orderings import lex
```

```
>>> vring("x,y,z", ZZ, lex)
Polynomial ring in x, y, z over ZZ with lex order
>>> x + y + z # noqa:
x + y + z
>>> type(_)
<class 'sympy.polys.rings.PolyElement'>
```

`sympy.polys.rings.`**`sring`**(*exprs, *symbols, **options*)

Construct a ring deriving generators and domain from options and input expressions.

Parameters

exprs : `Expr` (page 947) or sequence of `Expr` (page 947) (sympifiable)

symbols : sequence of `Symbol` (page 976)/`Expr` (page 947)

options : keyword arguments understood by `Options` (page 2642)

Examples

```
>>> from sympy import sring, symbols
```

```
>>> x, y, z = symbols("x,y,z")
>>> R, f = sring(x + 2*y + 3*z)
>>> R
Polynomial ring in x, y, z over ZZ with lex order
>>> f
x + 2*y + 3*z
>>> type(_)
<class 'sympy.polys.rings.PolyElement'>
```

class `sympy.polys.rings.`**`PolyRing`**(*symbols, domain, order=LexOrder()*)

Multivariate distributed polynomial ring.

add(**objs*)

Add a sequence of polynomials or containers of polynomials.

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
```

```
>>> R, x = ring("x", ZZ)
>>> R.add([ x**2 + 2*i + 3 for i in range(4) ])
4*x**2 + 24
```

(continues on next page)

(continued from previous page)

```
>>> _.factor_list()
(4, [(x**2 + 6, 1)])
```

add_gens(*symbols*)

Add the elements of symbols as generators to self

compose(*other*)

Add the generators of other to self

drop(**gens*)

Remove specified generators from this ring.

drop_to_ground(**gens*)

Remove specified generators from the ring and inject them into its domain.

index(*gen*)

Compute index of gen in self.gens.

monomial_basis(*i*)

Return the ith-basis element.

mul(**objs*)

Multiply a sequence of polynomials or containers of polynomials.

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
```

```
>>> R, x = ring("x", ZZ)
>>> R.mul([ x**2 + 2*i + 3 for i in range(4) ])
x**8 + 24*x**6 + 206*x**4 + 744*x**2 + 945
>>> _.factor_list()
(1, [(x**2 + 3, 1), (x**2 + 5, 1), (x**2 + 7, 1), (x**2 + 9, 1)])
```

class sympy.polys.rings.**PolyElement**

Element of multivariate distributed polynomial ring.

almosteq(*p2, tolerance=None*)

Approximate equality test for polynomials.

cancel(*g*)

Cancel common factors in a rational function f/g.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> (2*x**2 - 2).cancel(x**2 - 2*x + 1)
(2*x + 2, x - 1)
```

coeff(*element*)

Returns the coefficient that stands next to the given monomial.

Parameters

element : PolyElement (with `is_monomial = True`) or 1

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
```

```
>>> _, x, y, z = ring("x,y,z", ZZ)
>>> f = 3*x**2*y - x*y*z + 7*z**3 + 23
```

```
>>> f.coeff(x**2*y)
3
>>> f.coeff(x*y)
0
>>> f.coeff(1)
23
```


coeffs(*order=None*)

Ordered list of polynomial coefficients.

Parameters

order : *MonomialOrder* (page 2431) or coercible, optional

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.orderings import lex, grlex
```

```
>>> _, x, y = ring("x, y", ZZ, lex)
>>> f = x*y**7 + 2*x**2*y**3
```

```
>>> f.coeffs()
[2, 1]
>>> f.coeffs(grlex)
[1, 2]
```

const()

Returns the constant coeffcient.

content()

Returns GCD of polynomial's coefficients.

copy()

Return a copy of polynomial self.

Polynomials are mutable; if one is interested in preserving a polynomial, and one plans to use inplace operations, one can copy the polynomial. This method makes a shallow copy.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.rings import ring
```

```
>>> R, x, y = ring('x, y', ZZ)
>>> p = (x + y)**2
>>> p1 = p.copy()
>>> p2 = p
>>> p[R.zero_monom] = 3
>>> p
x**2 + 2*x*y + y**2 + 3
>>> p1
x**2 + 2*x*y + y**2
>>> p2
x**2 + 2*x*y + y**2 + 3
```


degree(*x=None*)

The leading degree in x or the main variable.

Note that the degree of 0 is negative infinity (the SymPy object -oo).

degrees()

A tuple containing leading degrees in all variables.

Note that the degree of 0 is negative infinity (the SymPy object -oo)

diff(*x*)

Computes partial derivative in x.

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
```

```
>>> _, x, y = ring("x,y", ZZ)
>>> p = x + x**2*y**3
>>> p.diff(x)
2*x*y**3 + 1
```