# CONTENTS

1 Base Classes for Rings, Algebras and Fields  

2 Ideals  

3 Ring Morphisms  

4 Quotient Rings  

5 Fraction Fields  

6 Localization  

7 Ring Extensions  

8 Generic Data Structures and Algorithms for Rings  

9 Utilities  

10 Derivation  

11 Indices and Tables  

Python Module Index  

Index
1.1 Rings

This module provides the abstract base class `Ring` from which all rings in Sage (used to) derive, as well as a selection of more specific base classes.

**Warning:** Those classes, except maybe for the lowest ones like `Ring`, `CommutativeRing`, `Algebra` and `CommutativeAlgebra`, are being progressively deprecated in favor of the corresponding categories, which are more flexible, in particular with respect to multiple inheritance.

The class inheritance hierarchy is:

- `Ring`
  - `Algebra`
  - `CommutativeRing`
    - `NoetherianRing`
    - `CommutativeAlgebra`
    - `IntegralDomain`
      - `DedekindDomain`
      - `PrincipalIdealDomain`

Subclasses of `PrincipalIdealDomain` are

- `EuclideanDomain`
- `Field`
  - `FiniteField`

Some aspects of this structure may seem strange, but this is an unfortunate consequence of the fact that Cython classes do not support multiple inheritance. Hence, for instance, `Field` cannot be a subclass of both `NoetherianRing` and `PrincipalIdealDomain`, although all fields are Noetherian PIDs.

(A distinct but equally awkward issue is that sometimes we may not know in advance whether or not a ring belongs in one of these classes; e.g. some orders in number fields are Dedekind domains, but others are not, and we still want to offer a unified interface, so orders are never instances of the `DedekindDomain` class.)

**AUTHORS:**

- David Harvey (2006-10-16): changed `CommutativeAlgebra` to derive from `CommutativeRing` instead of from `Algebra`. 
class sage.rings.ring.Algebra

Bases: Ring

Generic algebra

characteristic()

Return the characteristic of this algebra, which is the same as the characteristic of its base ring.

See objects with the base_ring attribute for additional examples. Here are some examples that explicitly use the Algebra class.

EXAMPLES:

```
sage: # needs sage.modules
sage: A = Algebra(ZZ); A
<sage.rings.ring.Algebra object at ...>
sage: A.characteristic()
0
sage: A = Algebra(GF(7^3, 'a'))
# needs sage.rings.finite_rings
sage: A.characteristic()  # needs sage.rings.finite_rings
7
```

has_standard_involution()

Return True if the algebra has a standard involution and False otherwise. This algorithm follows Algorithm 2.10 from John Voight’s Identifying the Matrix Ring. Currently the only type of algebra this will work for is a quaternion algebra. Though this function seems redundant, once algebras have more functionality, in particular have a method to construct a basis, this algorithm will have more general purpose.

EXAMPLES:

```
sage: # needs sage.combinat sage.modules
sage: B = QuaternionAlgebra(2)
sage: B.has_standard_involution()
True
sage: R.<x> = PolynomialRing(QQ)
sage: K.<u> = NumberField(x**2 - 2)  # needs sage.rings.number_field
sage: A = QuaternionAlgebra(K, -2, 5)  # needs sage.rings.number_field
sage: A.has_standard_involution()  # needs sage.rings.number_field
True
sage: L.<a,b> = FreeAlgebra(QQ, 2)
sage: L.has_standard_involution()
Traceback (most recent call last):
  ... Not Implemented Error: has_standard_involution is not implemented for this algebra
```
class sage.rings.ring.CommutativeAlgebra
    Bases: CommutativeRing
    
    Generic commutative algebra

    is_commutative()
    Return True since this algebra is commutative.

    EXAMPLES:
    Any commutative ring is a commutative algebra over itself:

    sage: A = sage.rings.ring.CommutativeAlgebra
    sage: A(ZZ).is_commutative()
    True
    sage: A(QQ).is_commutative()
    True

    Trying to create a commutative algebra over a non-commutative ring will result in a TypeError.

class sage.rings.ring.CommutativeRing
    Bases: Ring
    
    Generic commutative ring.

    derivation(arg=None, twist=None)
    Return the twisted or untwisted derivation over this ring specified by arg.

    Note: A twisted derivation with respect to \( \theta \) (or a \( \theta \)-derivation for short) is an additive map \( d \) satisfying the following axiom for all \( x, y \) in the domain:

    \[
    d(xy) = \theta(x)d(y) + d(x)y.
    \]

    INPUT:
    • arg – (optional) a generator or a list of coefficients that defines the derivation
    • twist – (optional) the twisting homomorphism

    EXAMPLES:
    sage: R.<x,y,z> = QQ[]
    sage: R.derivation()  # needs sage.modules
    d/dx

    In that case, arg could be a generator:

    sage: R.derivation(y)  # needs sage.modules
    d/dy

    or a list of coefficients:

    sage: R.derivation([1,2,3])  # needs sage.modules
    d/dx + 2*d/dy + 3*d/dz
It is not possible to define derivations with respect to a polynomial which is not a variable:

```
sage: R.derivation(x^2)  # needs sage.modules
Traceback (most recent call last):
... ValueError: unable to create the derivation
```

Here is an example with twisted derivations:

```
sage: R.<x,y,z> = QQ[]
sage: theta = R.hom([x^2, y^2, z^2])
sage: f = R.derivation(twist=theta); f
# needs sage.modules
0
sage: f.parent()
# needs sage.modules
Module of twisted derivations over Multivariate Polynomial Ring in x, y, z
over Rational Field (twisting morphism: x |--> x^2, y |--> y^2, z |--> z^2)
```

Specifying a scalar, the returned twisted derivation is the corresponding multiple of $\theta - id$:

```
sage: R.derivation(1, twist=theta)
# needs sage.modules
[x |--> x^2, y |--> y^2, z |--> z^2] - id
sage: R.derivation(x, twist=theta)
# needs sage.modules
x*(x |--> x^2, y |--> y^2, z |--> z^2] - id)
```

```
```

**derivation_module**

*(codomain=None, twist=None)*

Returns the module of derivations over this ring.

**INPUT:**

- `codomain` – an algebra over this ring or a ring homomorphism whose domain is this ring or `None` (default: `None`); if it is a morphism, the codomain of derivations will be the codomain of the morphism viewed as an algebra over self through the given morphism; if `None`, the codomain will be this ring
- `twist` – a morphism from this ring to `codomain` or `None` (default: `None`); if `None`, the coercion map from this ring to codomain will be used

**Note:** A twisted derivation with respect to $\theta$ (or a $\theta$-derivation for short) is an additive map $d$ satisfying the following axiom for all $x, y$ in the domain:

$$d(xy) = \theta(x)d(y) + d(x)y.$$
We can specify a different codomain:

```
sage: K = R.fraction_field()
sage: M = R.derivation_module(K); M
```

```
Module of derivations from Multivariate Polynomial Ring in x, y, z over Rational Field
to Fraction Field of Multivariate Polynomial Ring in x, y, z over Rational Field
```

```
sage: M.gen() / x
```

```
1/x*d/dx
```

Here is an example with a non-canonical defining morphism:

```
sage: ev = R.hom([QQ(0), QQ(1), QQ(2)])
sage: ev
```

```
Ring morphism:
From: Multivariate Polynomial Ring in x, y, z over Rational Field
To: Rational Field
Defn: x |--> 0
      y |--> 1
      z |--> 2
```

```
sage: M = R.derivation_module(ev)
sage: M
```

```
Module of derivations from Multivariate Polynomial Ring in x, y, z over Rational Field
to Rational Field
```

Elements in $M$ acts as derivations at $(0,1,2)$:

```
sage: Dx = M.gen(0); Dx
d/dx
sage: Dy = M.gen(1); Dy
d/dy
sage: Dz = M.gen(2); Dz
d/dz
sage: f = x^2 + y^2 + z^2
sage: Dx(f) # = 2*x evaluated at (0,1,2)
0
sage: Dy(f) # = 2*y evaluated at (0,1,2)
2
sage: Dz(f) # = 2*z evaluated at (0,1,2)
4
```

An example with a twisting homomorphism:
\begin{verbatim}
sage: theta = R.hom([x^2, y^2, z^2])
sage: M = R.derivation_module(twist=theta);
     M

Module of twisted derivations over Multivariate Polynomial Ring in x, y, z
over Rational Field (twisting morphism: x |--> x^2, y |--> y^2, z |--> z^2)
See also:
derivation()
extension(poly, name=None, names=None, **kwds)
Algebraically extends self by taking the quotient self[x] / (f(x)).
INPUT:
• poly – A polynomial whose coefficients are coercible into self
• name – (optional) name for the root of f
Note: Using this method on an algebraically complete field does not return this field; the construction self[x] / (f(x)) is done anyway.

EXAMPLES:
\begin{verbatim}
sage: R = QQ['x']
sage: y = polygen(R)
sage: R.extension(y^2 - 5, 'a')
Univariate Quotient Polynomial Ring in a over
Univariate Polynomial Ring in x over Rational Field with modulus a^2 - 5
\end{verbatim}
\begin{verbatim}
sage: # needs sage.rings.finite_rings
sage: P.<x> = PolynomialRing(GF(5))
sage: F.<a> = GF(5).extension(x^2 - 2)
sage: P.<t> = F[]
sage: R.<b> = F.extension(t^2 - a); R
Univariate Quotient Polynomial Ring in b over
Finite Field in a of size 5^2 with modulus b^2 + 4*a
\end{verbatim}
\end{verbatim}
\end{verbatim}
fraction_field()
Return the fraction field of self.
EXAMPLES:
\begin{verbatim}
sage: R = Integers(389)['x,y']
sage: Frac(R)
Fraction Field of Multivariate Polynomial Ring in x, y over Ring of integers
modulo 389
sage: R.fraction_field()
Fraction Field of Multivariate Polynomial Ring in x, y over Ring of integers
modulo 389
\end{verbatim}
frobenius_endomorphism(n=1)
INPUT:
\end{verbatim}
\end{verbatim}
• $n$ – a nonnegative integer (default: 1)

OUTPUT:
The $n$-th power of the arithmetic Frobenius endomorphism on this finite field.

EXAMPLES:

```
sage: K.<u> = PowerSeriesRing(GF(5))
sage: Frob = K.frobenius_endomorphism(); Frob
Frobenius endomorphism x |--> x^5 of Power Series Ring in u
over Finite Field of size 5
sage: Frob(u)
u^5
```

We can specify a power:

```
sage: f = K.frobenius_endomorphism(2); f
Frobenius endomorphism x |--> x^(5^2) of Power Series Ring in u
over Finite Field of size 5
sage: f(1+u)
1 + u^25
```

```
ideal_monoid()
Return the monoid of ideals of this ring.

EXAMPLES:

```
sage: ZZ.ideal_monoid()
Monoid of ideals of Integer Ring
sage: R.<x>=QQ[]; R.ideal_monoid()
Monoid of ideals of Univariate Polynomial Ring in x over Rational Field
```

is_commutative()
Return True, since this ring is commutative.

EXAMPLES:

```
sage: QQ.is_commutative()
True
sage: ZpCA(7).is_commutative() # needs sage.rings.padics
True
```

```
krull_dimension()
Return the Krull dimension of this commutative ring.

The Krull dimension is the length of the longest ascending chain of prime ideals.

localization(additional_units, names=None, normalize=True, category=None)
Return the localization of self at the given additional units.
```
EXAMPLES:

```python
sage: R.<x, y> = GF(3)[]

sage: R.localization((x*y, x**2 + y**2))  # needs sage.rings.finite_rings
Multivariate Polynomial Ring in x, y over Finite Field of size 3 localized at (y, x, x^2 + y^2)

sage: ~y in _  # needs sage.rings.finite_rings
True
```

```python
class sage.rings.ring.DedekindDomain

Bases: IntegralDomain

Generic Dedekind domain class.

A Dedekind domain is a Noetherian integral domain of Krull dimension one that is integrally closed in its field of fractions.

This class is deprecated, and not actually used anywhere in the Sage code base. If you think you need it, please create a category DedekindDomains, move the code of this class there, and use it instead.

integral_closure()

Return self since Dedekind domains are integrally closed.

EXAMPLES:

```python
sage: # needs sage.rings.number_field
sage: x = polygen(ZZ, 'x')

sage: K = NumberField(x**2 + 1, 's')

sage: OK = K.ring_of_integers()

sage: OK.integral_closure()
Gaussian Integers in Number Field in s with defining polynomial x^2 + 1

sage: OK.integral_closure() == OK
True

sage: QQ.integral_closure() == QQ
True
```

is_integrally_closed()

Return True since Dedekind domains are integrally closed.

EXAMPLES:

The following are examples of Dedekind domains (Noetherian integral domains of Krull dimension one that are integrally closed over its field of fractions).

```python
sage: ZZ.is_integrally_closed()
True

sage: x = polygen(ZZ, 'x')

sage: K = NumberField(x**2 + 1, 's')  # needs sage.rings.number_field

sage: OK = K.ring_of_integers()  # needs sage.rings.number_field

sage: OK.is_integrally_closed()  # needs sage.rings.number_field
```

(continues on next page)
These, however, are not Dedekind domains:

```
sage: QQ.is_integrally_closed()
True
sage: S = ZZ[sqrt(5)]; S.is_integrally_closed()
# needs sage.rings.number_field sage.symbolic
False
sage: T.<x,y> = PolynomialRing(QQ, 2); T
Multivariate Polynomial Ring in x, y over Rational Field
sage: T.is_integral_domain()
True
```

`is_noetherian()`

Return `True` since Dedekind domains are Noetherian.

**EXAMPLES:**
The integers, \( \mathbb{Z} \), and rings of integers of number fields are Dedekind domains:

```
sage: ZZ.is_noetherian()
True
sage: x = polygen(ZZ, 'x')
sage: K = NumberField(x^2 + 1, 's')
# needs sage.rings.number_field
sage: OK = K.ring_of_integers()
# needs sage.rings.number_field
sage: OK.is_noetherian()
# needs sage.rings.number_field
True
sage: QQ.is_noetherian()
True
```

`krull_dimension()`

Return 1 since Dedekind domains have Krull dimension 1.

**EXAMPLES:**
The following are examples of Dedekind domains (Noetherian integral domains of Krull dimension one that are integrally closed over its field of fractions):

```
sage: ZZ.krull_dimension()
1
sage: x = polygen(ZZ, 'x')
sage: K = NumberField(x^2 + 1, 's')
# needs sage.rings.number_field
sage: OK = K.ring_of_integers()
# needs sage.rings.number_field
sage: OK.krull_dimension()
# needs sage.rings.number_field
1
```

The following are not Dedekind domains but have a `krull_dimension` function:
```python
sage: QQ.krull_dimension()
0
sage: T.<x,y> = PolynomialRing(QQ,2); T
Multivariate Polynomial Ring in x, y over Rational Field
sage: T.krull_dimension()
2
sage: U.<x,y,z> = PolynomialRing(ZZ,3); U
Multivariate Polynomial Ring in x, y, z over Integer Ring
sage: U.krull_dimension()
4

sage: # needs sage.rings.number_field
sage: K.<i> = QuadraticField(-1)
sage: R = K.order(2*i); R
Order in Number Field in i with defining polynomial x^2 + 1 with i = 1*I
sage: R.is_maximal()
False
sage: R.krull_dimension()
1
```

**class** `sage.rings.ring.EuclideanDomain`

Bases: `PrincipalIdealDomain`

Generic Euclidean domain class.

This class is deprecated. Please use the `EuclideanDomains` category instead.

**parameter()**

Return an element of degree 1.

**EXAMPLES:**

```python
sage: R.<x>=QQ[]
sage: R.parameter()
x
```

**class** `sage.rings.ring.Field`

Bases: `PrincipalIdealDomain`

Generic field

**algebraic_closure()**

Return the algebraic closure of `self`.

**Note:** This is only implemented for certain classes of field.

**EXAMPLES:**

```python
sage: K = PolynomialRing(QQ,'x').fraction_field(); K
Fraction Field of Univariate Polynomial Ring in x over Rational Field
sage: K.algebraic_closure()
Traceback (most recent call last):
  ...NotImplementedError: Algebraic closures of general fields not implemented.
```
divides(x, y, coerce=True)
Return True if x divides y in this field (usually True in a field!). If coerce is True (the default), first coerce x and y into self.

EXAMPLES:
```
sage: QQ.divides(2, 3/4)
True
sage: QQ.divides(0, 5)
False
```

fraction_field()
Return the fraction field of self.

EXAMPLES:
Since fields are their own field of fractions, we simply get the original field in return:
```
sage: QQ.fraction_field()
Rational Field
sage: RR.fraction_field()  # needs sage.rings.real_mpfr
Real Field with 53 bits of precision
sage: CC.fraction_field()  # needs sage.rings.real_mpfr
Complex Field with 53 bits of precision
```

ideal(*gens, **kwds)
Return the ideal generated by gens.

EXAMPLES:
```
sage: QQ.ideal(2)
Principal ideal (1) of Rational Field
sage: QQ.ideal(0)
Principal ideal (0) of Rational Field
```

integral_closure()
Return this field, since fields are integrally closed in their fraction field.

EXAMPLES:
```
sage: QQ.integral_closure()
Rational Field
sage: Frac(ZZ['x,y']).integral_closure()  # needs sage.rings.number_field
Fraction Field of Multivariate Polynomial Ring in x, y over Integer Ring
```

is_field(proof=True)
Return True since this is a field.
is_integrally_closed()  
Return True since fields are trivially integrally closed in their fraction field (since they are their own fraction field).

EXAMPLES:

```
sage: Frac(ZZ['x,y']).is_integrally_closed()
True
```

is_noetherian()  
Return True since fields are Noetherian rings.

EXAMPLES:

```
sage: QQ.is_noetherian()
True
```

krull_dimension()  
Return the Krull dimension of this field, which is 0.

EXAMPLES:

```
sage: QQ.krull_dimension()
0
sage: Frac(QQ['x,y']).krull_dimension()
0
```

prime_subfield()  
Return the prime subfield of self.

EXAMPLES:

```
sage: k = GF(9, 'a')  # needs sage.rings.finite_rings
sage: k.prime_subfield()  # needs sage.rings.finite_rings
Finite Field of size 3
```

class sage.rings.ring.IntegralDomain  
Bases: CommutativeRing

Generic integral domain class.

This class is deprecated. Please use the sage.categories.integral_domains.IntegralDomains category instead.

is_field(proof=True)  
Return True if this ring is a field.

EXAMPLES:

```
sage: GF(7).is_field()
True
```
The following examples have their own `is_field` implementations:

```
sage: ZZ.is_field(); QQ.is_field()
False
True
sage: R.<x> = PolynomialRing(QQ); R.is_field()
False
```

`is_integral_domain(proof=True)`

Return True, since this ring is an integral domain.

(This is a naive implementation for objects with type `IntegralDomain`)

EXAMPLES:

```
sage: ZZ.is_integral_domain()
True
sage: QQ.is_integral_domain()
True
sage: ZZ['x'].is_integral_domain()
True
sage: R = ZZ.quotient(ZZ.ideal(10)); R.is_integral_domain()
False
```

`is_integrally_closed()`

Return True if this ring is integrally closed in its field of fractions; otherwise return False.

When no algorithm is implemented for this, then this function raises a `NotImplementedError`.

Note that `is_integrally_closed` has a naive implementation in fields. For every field $F$, $F$ is its own field of fractions, hence every element of $F$ is integral over $F$.

EXAMPLES:

```
sage: ZZ.is_integrally_closed()
True
sage: QQ.is_integrally_closed()
True
sage: QQbar.is_integrally_closed()
# needs sage.rings.number_field
True
sage: GF(5).is_integrally_closed()
True
sage: Z5 = Integers(5); Z5
Ring of integers modulo 5
sage: Z5.is_integrally_closed()
Traceback (most recent call last):
  ...
AttributeError: 'IntegerModRing_generic_with_category' object has no attribute 'is_integrally_closed'...
```

class sage.rings.ring.NoetherianRing

Bases: CommutativeRing

Generic Noetherian ring class.

A Noetherian ring is a commutative ring in which every ideal is finitely generated.
This class is deprecated, and not actually used anywhere in the Sage code base. If you think you need it, please create a category `NoetherianRings`, move the code of this class there, and use it instead.

**is_noetherian()**

Return `True` since this ring is Noetherian.

```
sage: ZZ.is_noetherian()
True
sage: QQ.is_noetherian()
True
sage: R.<x> = PolynomialRing(QQ)
```

```
sage: R.is_noetherian()
True
```

**class sage.rings.ring.PrincipalIdealDomain**

Bases: `IntegralDomain`

Generic principal ideal domain.

This class is deprecated. Please use the `PrincipalIdealDomains` category instead.

**class_group()**

Return the trivial group, since the class group of a PID is trivial.

```
sage: QQ.class_group()

# needs sage.groups
Trivial Abelian group
```

**content(x, y, coerce=True)**

Return the content of `x` and `y`, i.e. the unique element `c` of `self` such that `x/c` and `y/c` are coprime and integral.

```
sage: QQ.content(ZZ(42), ZZ(48)); type(QQ.content(ZZ(42), ZZ(48)))
6 <class 'sage.rings.rational.Rational'>
sage: QQ.content(1/2, 1/3)
1/6
sage: factor(1/2); factor(1/3); factor(1/6)
2^-1
3^-1
2^-1 * 3^-1
sage: a = (2*3)/(7*11); b = (13*17)/(19*23)
sage: factor(a); factor(b); factor(QQ.content(a,b))
2 * 3 * 7^-1 * 11^-1
13 * 17 * 19^-1 * 23^-1
7^-1 * 11^-1 * 19^-1 * 23^-1
```

Note the changes to the second entry:
sage: c = (2*3)/(7*11); d = (13*17)/(7*19*23)
sage: factor(c); factor(d); factor(QQ.content(c,d))
2 * 3 * 7^-1 * 11^-1  
7^-1 * 13 * 17 * 19^-1 * 23^-1  
7^-1 * 11^-1 * 19^-1 * 23^-1  
sage: e = (2*3)/(7*11); f = (13*17)/(7^3*19*23)
sage: factor(e); factor(f); factor(QQ.content(e,f))
2 * 3 * 7^-1 * 11^-1  
7^-3 * 13 * 17 * 19^-1 * 23^-1  
7^-3 * 11^-1 * 19^-1 * 23^-1

**gcd**(*x*, *y*, *coerce*=*True*)

Return the greatest common divisor of *x* and *y*, as elements of **self**.

**EXAMPLES:**

The integers are a principal ideal domain and hence a GCD domain:

sage: ZZ.gcd(42, 48)
6
sage: 42.factor(); 48.factor()
2 * 3 * 7
2^4 * 3
sage: ZZ.gcd(2^4*7^2*11, 2^3*11*13)
88
sage: 88.factor()
2^3 * 11

In a field, any nonzero element is a GCD of any nonempty set of nonzero elements. In previous versions, Sage used to return 1 in the case of the rational field. However, since github issue #10771, the rational field is considered as the fraction field of the integer ring. For the fraction field of an integral domain that provides both GCD and LCM, it is possible to pick a GCD that is compatible with the GCD of the base ring:

sage: QQ.gcd(ZZ(42), ZZ(48)); type(QQ.gcd(ZZ(42), ZZ(48)))
6
<class 'sage.rings.rational.Rational'>
sage: QQ.gcd(1/2, 1/3)
1/6

Polynomial rings over fields are GCD domains as well. Here is a simple example over the ring of polynomials over the rationals as well as over an extension ring. Note that gcd requires x and y to be coercible:

sage: # needs sage.rings.number_field
sage: R.<x> = PolynomialRing(QQ)
sage: S.<a> = NumberField(x^2 - 2, 'a')
sage: f = (x - a)*(x + a); g = (x - a)*(x^2 - 2)
sage: print(f); print(g)
x^2 - 2
x^3 - a*x^2 - 2*x + 2*a
sage: f in R
True
sage: g in R
False

(continues on next page)
sage: R.gcd(f, g)
Traceback (most recent call last):
...
TypeError: Unable to coerce 2*a to a rational
sage: R.base_extend(S).gcd(f,g)
x^2 - 2
sage: R.base_extend(S).gcd(f, (x - a)*(x^2 - 3))
x - a

is_noetherian()

Every principal ideal domain is noetherian, so we return True.

EXAMPLES:
sage: Zp(5).is_noetherian()          # needs sage.rings.padics
True

class sage.rings.ring.Ring

Bases: ParentWithGens

Generic ring class.

base_extend(R)

EXAMPLES:
sage: QQ.base_extend(GF(7))
Traceback (most recent call last):
...
TypeError: no base extension defined
sage: ZZ.base_extend(GF(7))
Finite Field of size 7

category()

Return the category to which this ring belongs.

Note: This method exists because sometimes a ring is its own base ring. During initialisation of a ring \( R \), it may be checked whether the base ring (hence, the ring itself) is a ring. Hence, it is necessary that \( R . category() \) tells that \( R \) is a ring, even before its category is properly initialised.

EXAMPLES:
sage: FreeAlgebra(QQ, 3, 'x').category()  # todo: use a ring which is not an algebra!
Category of algebras with basis over Rational Field

Since a quotient of the integers is its own base ring, and during initialisation of a ring it is tested whether the base ring belongs to the category of rings, the following is an indirect test that the category() method of rings returns the category of rings even before the initialisation was successful:
sage: I = Integers(15)
sage: I.base_ring() is I
True

\texttt{sage: I.category()}
Join of Category of finite commutative rings
and Category of subquotients of monoids
and Category of quotients of semigroups
and Category of finite enumerated sets

\textbf{epsilon()}

Return the precision error of elements in this ring.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: RDF.epsilon()
2.220446049250313e-16
sage: ComplexField(53).epsilon()
                # needs sage.rings.real_mpfr
2.22044604925031e-16
sage: RealField(10).epsilon()
               # needs sage.rings.real_mpfr
0.0020
\end{verbatim}

For exact rings, zero is returned:

\begin{verbatim}
sage: ZZ.epsilon()
0
\end{verbatim}

This also works over derived rings:

\begin{verbatim}
sage: RR['x'].epsilon()
                # needs sage.rings.real_mpfr
2.22044604925031e-16
sage: QQ['x'].epsilon()
0
\end{verbatim}

For the symbolic ring, there is no reasonable answer:

\begin{verbatim}
sage: SR.epsilon()
                # needs sage.symbolic
Traceback (most recent call last):
  ...
NotImplementedError
\end{verbatim}

\textbf{ideal(*args, **kwds)}

Return the ideal defined by \texttt{x}, i.e., generated by \texttt{x}.

\textbf{INPUT:}

- \texttt{*x} – list or tuple of generators (or several input arguments)
- \texttt{coerce} – bool (default: \texttt{True}); this must be a keyword argument. Only set it to \texttt{False} if you are certain that each generator is already in the ring.
- \texttt{ideal_class} – callable (default: \texttt{self._ideal_class()}); this must be a keyword argument. A constructor for ideals, taking the ring as the first argument and then the generators. Usually a subclass of \texttt{Ideal_generic} or \texttt{Ideal_nc}. 
Further named arguments (such as \texttt{side} in the case of non-commutative rings) are forwarded to the ideal class.

**EXAMPLES:**

```plaintext
sage: R.<x,y> = QQ[]
sage: R.ideal(x,y)
Ideal (x, y) of Multivariate Polynomial Ring in x, y over Rational Field
sage: R.ideal(x+y^2)
Ideal (y^2 + x) of Multivariate Polynomial Ring in x, y over Rational Field
sage: R.ideal([x^3,y^3+x^3])
Ideal (x^3, x^3 + y^3) of Multivariate Polynomial Ring in x, y over Rational Field
```

Here is an example over a non-commutative ring:

```plaintext
sage: A = SteenrodAlgebra(2)
# needs sage.combinat sage.modules
sage: A.ideal(A.1, A.2^2)
Twosided Ideal (Sq(2), Sq(2,2)) of mod 2 Steenrod algebra, milnor basis
sage: A.ideal(A.1, A.2^2, side='left')
Left Ideal (Sq(2), Sq(2,2)) of mod 2 Steenrod algebra, milnor basis
```

\textbf{ideal_monoid()}

Return the monoid of ideals of this ring.

**EXAMPLES:**

```plaintext
sage: # needs sage.combinat sage.modules
sage: F.<x,y,z> = FreeAlgebra(ZZ, 3)
sage: I = F * [x*y + y*z, x^2 + x*y - y*x - y^2] * F
sage: Q = F.quotient(I)
sage: Q.ideal_monoid()
Monoid of ideals of Quotient of Free Algebra on 3 generators (x, y, z)
over Integer Ring by the ideal (x*y + y*z, x^2 + x*y - y*x - y^2)
sage: F.<x,y,z> = FreeAlgebra(ZZ, implementation='letterplace')
sage: I = F * [x*y + y*z, x^2 + x*y - y*x - y^2] * F
sage: Q = F.quotient(I)
sage: Q.ideal_monoid()
Monoid of ideals of Quotient of Free Associative Unital Algebra
on 3 generators (x, y, z) over Integer Ring
by the ideal (x*y + y*z, x*x + x*y - y*x - y*y)
```

\textbf{is_commutative()}

Return True if this ring is commutative.

**EXAMPLES:**

```plaintext
sage: QQ.is_commutative()
True
sage: QQ['x,y,z'].is_commutative()
True
sage: Q.<i,j,k> = QuaternionAlgebra(QQ, -1, -1)
#...
```
is_exact()
Return True if elements of this ring are represented exactly, i.e., there is no precision loss when doing arithmetic.

Note: This defaults to True, so even if it does return True you have no guarantee (unless the ring has properly overloaded this).

EXAMPLES:

```
sage: QQ.is_exact()  # indirect doctest
True
sage: ZZ.is_exact()
True
sage: Qp(7).is_exact()  # needs sage.rings.padics
False
sage: Zp(7, type='capped-abs').is_exact()  # needs sage.rings.padics
False
```

is_field(proof=True)
Return True if this ring is a field.

INPUT:

• proof – (default: True) Determines what to do in unknown cases

ALGORITHM:
If the parameter proof is set to True, the returned value is correct but the method might throw an error. Otherwise, if it is set to False, the method returns True if it can establish that self is a field and False otherwise.

EXAMPLES:

```
sage: QQ.is_field()
True
sage: GF(9, 'a').is_field()  # needs sage.rings.finite_rings
True
sage: ZZ.is_field()
False
sage: QQ['x'].is_field()
False
sage: Frac(QQ['x']).is_field()
True
```

This illustrates the use of the proof parameter:
is_integral_domain(proof=True)

Return True if this ring is an integral domain.

INPUT:

• proof – (default: True) Determines what to do in unknown cases

ALGORITHM:

If the parameter proof is set to True, the returned value is correct but the method might throw an error. Otherwise, if it is set to False, the method returns True if it can establish that self is an integral domain and False otherwise.

EXAMPLES:

```
sage: QQ.is_integral_domain()
True
sage: ZZ.is_integral_domain()
True
sage: ZZ['x,y,z'].is_integral_domain()
True
sage: Integers(8).is_integral_domain()
False
sage: Zp(7).is_integral_domain()  # needs sage.rings.padics
True
sage: Qp(7).is_integral_domain()  # needs sage.rings.padics
True
sage: R.<a,b> = ZZ[]
... needs sage.libs.singular
sage: S.<x,y> = R.quo((b^3))
... needs sage.libs.singular
sage: S.is_integral_domain()
False
```

This illustrates the use of the proof parameter:

```
sage: R.<a,b> = ZZ[
... needs sage.libs.singular
sage: S.<x,y> = R.quo((b^3))
... needs sage.libs.singular
sage: S.is_integral_domain(proof=True)
... needs sage.libs.singular
False
```

(continues on next page)
Traceback (most recent call last):
  ... 
NotImplementedError
sage: S.is_integral_domain(proof=False) # needs sage.libs.singular
False

is_noetherian()
Return True if this ring is Noetherian.

EXAMPLES:

sage: QQ.is_noetherian()
True
sage: ZZ.is_noetherian()
True

is_prime_field()
Return True if this ring is one of the prime fields \( \mathbb{Q} \) or \( \mathbb{F}_p \).

EXAMPLES:

sage: QQ.is_prime_field()
True
sage: GF(3).is_prime_field()
True
sage: GF(9, 'a').is_prime_field() # needs sage.rings.finite_rings
False
sage: ZZ.is_prime_field()
False
sage: QQ['x'].is_prime_field()
False
sage: Qp(19).is_prime_field() # needs sage.rings.padics
False

is_subring(other)
Return True if the canonical map from self to other is injective.
Raises a NotImplementedError if not known.

EXAMPLES:

sage: ZZ.is_subring(QQ)
True
sage: ZZ.is_subring(GF(19))
False

one()
Return the one element of this ring (cached), if it exists.

EXAMPLES:
The result is cached:

```python
sage: ZZ.one() is ZZ.one()
True
```

**order()**
The number of elements of self.

```python
sage: GF(19).order()
19
sage: QQ.order()
+Infinity
```

**principal_ideal**(gen, coerce=True)
Return the principal ideal generated by gen.

```python
sage: R.<x,y> = ZZ[]
sage: R.principal_ideal(x+2*y)
Ideal (x + 2*y) of Multivariate Polynomial Ring in x, y over Integer Ring
```

**random_element**(bound=2)
Return a random integer coerced into this ring, where the integer is chosen uniformly from the interval \([-\text{bound}, \text{bound}].
```

**unit_ideal()**
Return the unit ideal of this ring.

```python
sage: Zp(7).unit_ideal()  # needs sage.rings.padics
Principal ideal (1 + O(7^20)) of 7-adic Ring with capped relative precision 20
```

**zero()**
Return the zero element of this ring (cached).

```python
```
sage: ZZ.zero()
0
sage: QQ.zero()
0
sage: QQ['x'].zero()
0

The result is cached:

sage: ZZ.zero() is ZZ.zero()
True

zero_ideal()
Return the zero ideal of this ring (cached).

EXAMPLES:

sage: ZZ.zero_ideal()
Principal ideal (0) of Integer Ring
sage: QQ.zero_ideal()
Principal ideal (0) of Rational Field
sage: QQ['x'].zero_ideal()
Principal ideal (0) of Univariate Polynomial Ring in x over Rational Field

The result is cached:

sage: ZZ.zero_ideal() is ZZ.zero_ideal()
True

zeta(n=2, all=False)
Return a primitive $n$-th root of unity in self if there is one, or raise a ValueError otherwise.

INPUT:

• $n$ – positive integer
• all – bool (default: False) - whether to return a list of all primitive $n$-th roots of unity. If True, raise a ValueError if self is not an integral domain.

OUTPUT:

Element of self of finite order

EXAMPLES:

sage: QQ.zeta()
-1
sage: QQ.zeta(1)
1
sage: CyclotomicField(6).zeta(6)  # _
needs sage.rings.number_field
zeta6
sage: CyclotomicField(3).zeta(3)  # _
needs sage.rings.number_field
zeta3
sage: CyclotomicField(3).zeta(3).multiplicative_order()  # _
(continues on next page)
needs sage.rings.number_field
3
sage: # needs sage.rings.finite_rings
sage: a = GF(7).zeta(); a
3
sage: a.multiplicative_order()
6
sage: a = GF(49,'z').zeta(); a
z
sage: a.multiplicative_order()
48
sage: a = GF(49,'z').zeta(2); a
6
sage: a.multiplicative_order()
2
sage: QQ.zeta(3)
Traceback (most recent call last):
...
ValueError: no n-th root of unity in rational field
sage: Zp(7, prec=8).zeta()
# needs sage.rings.padics
3 + 4*7 + 6*7^2 + 3*7^3 + 2*7^5 + 6*7^6 + 2*7^7 + O(7^8)

zeta_order()

Return the order of the distinguished root of unity in self.

EXAMPLES:

sage: CyclotomicField(19).zeta_order()  # needs sage.rings.number_field
38
sage: GF(19).zeta_order()  
18
sage: GF(5^3,'a').zeta_order()  # needs sage.rings.finite_rings
124
sage: Zp(7, prec=8).zeta_order()  # needs sage.rings.padics
6

sage.rings.ring.is_Ring(x)

Return True if x is a ring.

EXAMPLES:

sage: from sage.rings.ring import is_Ring
sage: is_Ring(ZZ)
True
sage: MS = MatrixSpace(QQ, 2)
# needs sage.modules
sage: is_Ring(MS)
#
1.2 Abstract base classes for rings

class sage.rings.abc.AlgebraicField

Bases: AlgebraicField_common

Abstract base class for AlgebraicField.

This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

```
sage: import sage.rings.abc
sage: isinstance(QQbar, sage.rings.abc.AlgebraicField)  # needs sage.rings.number_field
True
sage: isinstance(AA, sage.rings.abc.AlgebraicField)  # needs sage.rings.number_field
False
```

By design, there is a unique direct subclass:

```
sage: sage.rings.abc.AlgebraicField.__subclasses__()  # needs sage.rings.number_field
[<class 'sage.rings.qqbar.AlgebraicField'>]
sage: len(sage.rings.abc.AlgebraicField.__subclasses__()) <= 1
True
```

class sage.rings.abc.AlgebraicField_common

Bases: Field

Abstract base class for AlgebraicField_common.

This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

```
sage: import sage.rings.abc
sage: isinstance(QQbar, sage.rings.abc.AlgebraicField_common)  # needs sage.rings.number_field
True
sage: isinstance(AA, sage.rings.abc.AlgebraicField_common)  # needs sage.rings.number_field
True
```

By design, other than the abstract subclasses AlgebraicField and AlgebraicRealField, there is only one direct implementation subclass:
class sage.rings.abc.AlgebraicField
Bases: AlgebraicField_common
Abstract base class for AlgebraicField.
This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

```sage
sage: import sage.rings.abc
sage: isinstance(QQbar, sage.rings.abc.AlgebraicField)  # needs sage.rings.number_field
False
sage: isinstance(AA, sage.rings.abc.AlgebraicField)  # needs sage.rings.number_field
True
```

By design, there is a unique direct subclass:

```sage
sage: len(sage.rings.abc.AlgebraicField.__subclasses__()) <= 3
True
```

class sage.rings.abc.AlgebraicRealField
Bases: AlgebraicField_common
Abstract base class for AlgebraicRealField.
This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

```sage
sage: import sage.rings.abc
sage: f = x.function(x).parent()  # needs sage.symbolic
sage: isinstance(f, sage.rings.abc.CallableSymbolicExpressionRing)  # needs sage.symbolic
True
```

By design, there is a unique direct subclass:

```sage
sage: len(sage.rings.abc.AlgebraicRealField.__subclasses__()) <= 1
True
```

class sage.rings.abc.CallableSymbolicExpressionRing
Bases: SymbolicRing
Abstract base class for CallableSymbolicExpressionRing_class.
This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

```sage
sage: import sage.rings.abc
sage: f = x.function(x).parent()  # needs sage.symbolic
sage: isinstance(f, sage.rings.abc.CallableSymbolicExpressionRing)  # needs sage.symbolic
True
```

By design, there is a unique direct subclass:

```sage
sage: len(sage.rings.abc.CallableSymbolicExpressionRing.__subclasses__()) <= 1
True
```
class sage.rings.abc.ComplexBallField

Bases: Field

Abstract base class for ComplexBallField.
This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

```python
sage: import sage.rings.abc
sage: isinstance(CBF, sage.rings.abc.ComplexBallField)  # needs sage.libs.flint
True
```

By design, there is a unique direct subclass:

```python
sage: sage.rings.abc.ComplexBallField.__subclasses__()  # needs sage.libs.flint
[<class 'sage.rings.complex_arb.ComplexBallField'>]
sage: len(sage.rings.abc.ComplexBallField.__subclasses__()) <= 1
True
```

class sage.rings.abc.ComplexDoubleField

Bases: Field

Abstract base class for ComplexDoubleField_class.
This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

```python
sage: import sage.rings.abc
sage: isinstance(CDF, sage.rings.abc.ComplexDoubleField)  # needs sage.rings.complex_double
True
```

By design, there is a unique direct subclass:

```python
sage: sage.rings.abc.ComplexDoubleField.__subclasses__()  # needs sage.rings.complex_double
[<class 'sage.rings.complex_double.ComplexDoubleField_class'>]
sage: len(sage.rings.abc.ComplexDoubleField.__subclasses__()) <= 1
True
```

class sage.rings.abc.ComplexField

Bases: Field

Abstract base class for ComplexField_class.
This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:
sage: import sage.rings.abc
sage: isinstance(CC, sage.rings.abc.ComplexField)  
# needs sage.rings.real_mpfr
True

By design, there is a unique direct subclass:

sage: sage.rings.abc.ComplexField.__subclasses__()  
# needs sage.rings.real_mpfr
[<class 'sage.rings.complex_mpfr.ComplexField_class'>]
sage: len(sage.rings.abc.ComplexField.__subclasses__()) <= 1
True

class sage.rings.abc.ComplexIntervalField

Bases: Field

Abstract base class for ComplexIntervalField_class.

This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

sage: import sage.rings.abc
sage: isinstance(CIF, sage.rings.abc.ComplexIntervalField)  
# needs sage.rings.complex_interval_field
True

By design, there is a unique direct subclass:

sage: sage.rings.abc.ComplexIntervalField.__subclasses__()  
# needs sage.rings.complex_interval_field
[<class 'sage.rings.complex_interval_field.ComplexIntervalField_class'>]
sage: len(sage.rings.abc.ComplexIntervalField.__subclasses__()) <= 1
True

class sage.rings.abc.IntegerModRing

Bases: object

Abstract base class for IntegerModRing_generic.

This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

sage: import sage.rings.abc
sage: isinstance(Integers(7), sage.rings.abc.IntegerModRing)  
True

By design, there is a unique direct subclass:

sage: sage.rings.abc.IntegerModRing.__subclasses__()  
[<class 'sage.rings.finite_rings.integer_mod_ring.IntegerModRing_generic'>]
sage: len(sage.rings.abc.IntegerModRing.__subclasses__()) <= 1
True
class sage.rings.abc.NumberField_cyclotomic
Bases: Field

Abstract base class for NumberField_cyclotomic.

This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

```
sage: import sage.rings.abc
sage: K.<zeta> = CyclotomicField(15)  # needs sage.rings.number_field
sage: isinstance(K, sage.rings.abc.NumberField_cyclotomic)  # needs sage.rings.number_field
True
```

By design, there is a unique direct subclass:

```
sage: len(sage.rings.abc.NumberField_cyclotomic.__subclasses__()) <= 1
True
```

class sage.rings.abc.NumberField_quadratic
Bases: Field

Abstract base class for NumberField_quadratic.

This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

```
sage: import sage.rings.abc
sage: K.<sqrt2> = QuadraticField(2)  # needs sage.rings.number_field
sage: isinstance(K, sage.rings.abc.NumberField_quadratic)  # needs sage.rings.number_field
True
```

By design, there is a unique direct subclass:

```
sage: len(sage.rings.abc.NumberField_quadratic.__subclasses__()) <= 1
True
```

class sage.rings.abc.Order
Bases: object

Abstract base class for Order.

This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:
sage: import sage.rings.abc
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 1); O = K.order(2*a)  # ← needs sage.rings.number_field
sage: isinstance(O, sage.rings.abc.Order)  # ← needs sage.rings.number_field
True

By design, there is a unique direct subclass:

sage: sage.rings.abc.Order.__subclasses__()
[<class 'sage.rings.number_field.order.Order'>]  # ← needs sage.rings.number_field

sage: len(sage.rings.abc.Order.__subclasses__()) <= 1
True

class sage.rings.abc.RealBallField
Bases: Field

Abstract base class for RealBallField.

This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

sage: import sage.rings.abc
sage: isinstance(RBF, sage.rings.abc.RealBallField)  # ← needs sage.libs.flint
True

By design, there is a unique direct subclass:

sage: sage.rings.abc.RealBallField.__subclasses__()
[<class 'sage.rings.real_arb.RealBallField'>]  # ← needs sage.libs.flint

sage: len(sage.rings.abc.RealBallField.__subclasses__()) <= 1
True

class sage.rings.abc.RealDoubleField
Bases: Field

Abstract base class for RealDoubleField_class.

This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

sage: import sage.rings.abc
sage: isinstance(RDF, sage.rings.abc.RealDoubleField)  # ← needs sage.libs.flint
True

By design, there is a unique direct subclass:
class sage.rings.abc.RealField

Bases: Field

Abstract base class for RealField_class.

This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

```python
sage: import sage.rings.abc
sage: isinstance(RR, sage.rings.abc.RealField)  # needs sage.rings.real_mpfr
True
```

By design, there is a unique direct subclass:

```python
sage: sage.rings.abc.RealField.__subclasses__()  # needs sage.rings.real_mpfr
[<class 'sage.rings.real_mpfr.RealField_class'>]
```

```python
sage: len(sage.rings.abc.RealField.__subclasses__()) <= 1
True
```

class sage.rings.abc.RealIntervalField

Bases: Field

Abstract base class for RealIntervalField_class.

This class is defined for the purpose of isinstance tests. It should not be instantiated.

EXAMPLES:

```python
sage: import sage.rings.abc
sage: isinstance(RIF, sage.rings.abc.RealIntervalField)  # needs sage.rings.real_interval_field
True
```

By design, there is a unique direct subclass:

```python
sage: sage.rings.abc.RealIntervalField.__subclasses__()  # needs sage.rings.real_interval_field
[<class 'sage.rings.real_mpfi.RealIntervalField_class'>]
```

```python
sage: len(sage.rings.abc.RealIntervalField.__subclasses__()) <= 1
True
```

class sage.rings.abc.SymbolicRing

Bases: CommutativeRing

Abstract base class for SymbolicRing.

This class is defined for the purpose of isinstance tests. It should not be instantiated.

1.2. Abstract base classes for rings 31
EXAMPLES:

```python
sage: import sage.rings.abc
sage: isinstance(SR, sage.rings.abc.SymbolicRing)  # needs sage.symbolic
True
```

By design, other than the abstract subclass `CallableSymbolicExpressionRing`, there is only one direct implementation subclass:

```python
sage: sage.rings.abc.SymbolicRing.__subclasses__()  # needs sage.symbolic
[<class 'sage.rings.abc.CallableSymbolicExpressionRing'>,
 <class 'sage.symbolic.ring.SymbolicRing'>]
```

```python
sage: len(sage.rings.abc.SymbolicRing.__subclasses__()) <= 2
True
```

```python
class sage.rings.abc.UniversalCyclotomicField
    Bases: Field

    Abstract base class for UniversalCyclotomicField. This class is defined for the purpose of `isinstance()` tests. It should not be instantiated.

    EXAMPLES:

    ```python
    sage: import sage.rings.abc
    sage: isinstance(Zp(5), sage.rings.abc.UniversalCyclotomicField)  # needs sage.libs.gap sage.rings.number_field
    False
    sage: isinstance(Qp(5), sage.rings.abc.UniversalCyclotomicField)  # needs sage.libs.gap sage.rings.number_field
    False
    ```
```

By design, there is a unique direct subclass:

```python
sage: sage.rings.abc.UniversalCyclotomicField.__subclasses__()  # needs sage.libs.gap sage.rings.number_field
[<class 'sage.rings.universal_cyclotomic_field.UniversalCyclotomicField'>]
```

```python
sage: len(sage.rings.abc.UniversalCyclotomicField.__subclasses__()) <= 1
True
```

```python
class sage.rings.abc.pAdicField
    Bases: Field

    Abstract base class for pAdicFieldGeneric.

    This class is defined for the purpose of `isinstance()` tests. It should not be instantiated.

    EXAMPLES:

    ```python
    sage: import sage.rings.abc
    sage: isinstance(Zp(5), sage.rings.abc.pAdicField)  # needs sage.rings.padics
    False
    sage: isinstance(Qp(5), sage.rings.abc.pAdicField)  # needs sage.rings.padics
    False
    ```
```

(continues on next page)
By design, there is a unique direct subclass:

```python
sage: sage.rings.abc.pAdicField.__subclasses__()  # needs sage.rings.padics
[<class 'sage.rings.padics.generic_nodes.pAdicFieldGeneric'>]
```

```python
sage: len(sage.rings.abc.pAdicField.__subclasses__()) <= 1
True
```

```
class sage.rings.abc.pAdicRing

    Bases: EuclideanDomain

    Abstract base class for pAdicRingGeneric.

    This class is defined for the purpose of isinstance tests. It should not be instantiated.

    EXAMPLES:

    ```
    sage: import sage.rings.abc
    sage: isinstance(Zp(5), sage.rings.abc.pAdicRing)  # needs sage.rings.padics
    True
    sage: isinstance(Qp(5), sage.rings.abc.pAdicRing)  # needs sage.rings.padics
    False
    ```
```

By design, there is a unique direct subclass:

```python
sage: sage.rings.abc.pAdicRing.__subclasses__()  # needs sage.rings.padics
[<class 'sage.rings.padics.generic_nodes.pAdicRingGeneric'>]
```

```python
sage: len(sage.rings.abc.pAdicRing.__subclasses__()) <= 1
True
```
2.1 Ideals of commutative rings

Sage provides functionality for computing with ideals. One can create an ideal in any commutative or non-commutative ring $R$ by giving a list of generators, using the notation $R.\text{ideal}([a,b,...])$. The case of non-commutative rings is implemented in $\text{noncommutative_ideals}$. A more convenient notation may be $R^*\{a,b,...\}$ or $\{a,b,...\}^*R$. If $R$ is non-commutative, the former creates a left and the latter a right ideal, and $R^*\{a,b,...\}*R$ creates a two-sided ideal.

sage.rings.ideal.Cyclic($R$, $n=None$, $\text{homog}=False$, $\text{singular}=None$)
Ideal of cyclic n-roots from 1-st n variables of $R$ if $R$ is coercible to $\text{Singular}$.

INPUT:
- $R$ – base ring to construct ideal for
- $n$ – number of cyclic roots (default: None). If None, then $n$ is set to $R.\text{ngens}()$.
- $\text{homog}$ – (default: False) if True a homogeneous ideal is returned using the last variable in the ideal
- $\text{singular}$ – singular instance to use

Note: $R$ will be set as the active ring in $\text{Singular}$

EXAMPLES:
An example from a multivariate polynomial ring over the rationals:

```
sage: P.<x,y,z> = PolynomialRing(QQ, 3, order='lex')
sage: I = sage.rings.ideal.Cyclic(P); I
Ideal (x + y + z, x*y + x*z + y*z, x*y*z - 1)
of Multivariate Polynomial Ring in x, y, z over Rational Field

sage: I.groebner_basis()
[\text{continues on next page}]
```

We compute a Groebner basis for cyclic 6, which is a standard benchmark and test ideal:

```
sage: R.<x,y,z,t,u,v> = QQ['x,y,z,t,u,v']
sage: I = sage.rings.ideal.Cyclic(R, 6)
```
(continues on next page)
sage: B = I.groebner_basis() # needs sage.libs.singular
sage: len(B) # needs sage.libs.singular
45

sage.rings.ideal.FieldIdeal(R)

Let \( q = R.\text{base\_ring()}.\text{order()} \) and \((x_0, ..., x_n) = R.\text{gens()}\) then if \( q \) is finite this constructor returns

\( \langle x_0^q - x_0, ..., x_n^q - x_n \rangle \).

We call this ideal the field ideal and the generators the field equations.

EXAMPLES:
The field ideal generated from the polynomial ring over two variables in the finite field of size 2:

sage: P.<x,y> = PolynomialRing(GF(2), 2)
sage: I = sage.rings.ideal.FieldIdeal(P); I
Ideal (x^2 + x, y^2 + y) of
Multivariate Polynomial Ring in x, y over Finite Field of size 2

Another, similar example:

sage: Q.<x1,x2,x3,x4> = PolynomialRing(GF(2^4, name='alpha'), 4) # needs sage.rings.finite_rings
sage: J = sage.rings.ideal.FieldIdeal(Q); J
Ideal (x1^16 + x1, x2^16 + x2, x3^16 + x3, x4^16 + x4) of
Multivariate Polynomial Ring in x1, x2, x3, x4
over Finite Field in alpha of size 2^4

sage.rings.ideal.Ideal(*args, **kwds)

Create the ideal in ring with given generators.

There are some shorthand notations for creating an ideal, in addition to using the Ideal() function:

- \( R.\text{ideal}(\text{gens}, \text{coerce=True}) \)
- \( \text{gens}^R \)
- \( R^*\text{gens} \)

INPUT:

- \( R \) - A ring (optional; if not given, will try to infer it from \text{gens})
- \( \text{gens} \) - list of elements generating the ideal
- \( \text{coerce} \) - bool (optional, default: True); whether \text{gens} need to be coerced into the ring.

OUTPUT: The ideal of ring generated by \text{gens}.

EXAMPLES:

sage: R.<x> = ZZ[]
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: I
Ideal \((x^2 + 3x + 4, x^2 + 1)\) of Univariate Polynomial Ring in \(x\) over Integer Ring

\[\text{sage: } \text{Ideal}(R, [4 + 3x + x^2, 1 + x^2])\]

Ideal \((x^2 + 3x + 4, x^2 + 1)\) of Univariate Polynomial Ring in \(x\) over Integer Ring

\[\text{sage: } \text{Ideal}((4 + 3x + x^2, 1 + x^2))\]

Ideal \((x^2 + 3x + 4, x^2 + 1)\) of Univariate Polynomial Ring in \(x\) over Integer Ring

\[\text{sage: } \text{ideal}(x^2-2x+1, x^2-1)\]

Ideal \((x^2 - 2x + 1, x^2 - 1)\) of Univariate Polynomial Ring in \(x\) over Integer Ring

\[\text{sage: } \text{ideal}([x^2-2x+1, x^2-1])\]

Ideal \((x^2 - 2x + 1, x^2 - 1)\) of Univariate Polynomial Ring in \(x\) over Integer Ring

\[\text{sage: } \text{ideal}(f^2 \text{ for } f \text{ in } 1)\]

Ideal \((x^4 - 4x^3 + 6x^2 - 4x + 1, x^4 - 2x^2 + 1)\) of Univariate Polynomial Ring in \(x\) over Integer Ring

This example illustrates how Sage finds a common ambient ring for the ideal, even though 1 is in the integers (in this case).

\[\text{sage: } R.<t> = ZZ['t']\]
\[\text{sage: } i = \text{ideal}(1,t,t^2)\]
\[\text{sage: } i\]

Ideal \((1, t, t^2)\) of Univariate Polynomial Ring in \(t\) over Integer Ring

\[\text{sage: } \text{ideal}(1/2,t,t^2)\]

Principal ideal \((1)\) of Univariate Polynomial Ring in \(t\) over Rational Field

This shows that the issues at github issue #1104 are resolved:

\[\text{sage: } \text{Ideal}(3, 5)\]

Principal ideal \((1)\) of Integer Ring

\[\text{sage: } \text{Ideal}(ZZ, 3, 5)\]

Principal ideal \((1)\) of Integer Ring

\[\text{sage: } \text{Ideal}(2, 4, 6)\]

Principal ideal \((2)\) of Integer Ring

You have to provide enough information that Sage can figure out which ring to put the ideal in.

\[\text{sage: } I = \text{Ideal}([])\]

Traceback (most recent call last):
...
ValueError: unable to determine which ring to embed the ideal in

\[\text{sage: } I = \text{Ideal}()\]

Traceback (most recent call last):
...
ValueError: need at least one argument

Note that some rings use different ideal implementations than the standard, even if they are PIDs:

\[\text{sage: } R.<x> = GF(5)[]\]
\[\text{sage: } I = R * (x^2 + 3)\]
\[\text{sage: } \text{type}(I)\]

<class 'sage.rings.polynomial.ideal.Ideal_1poly_field'>

2.1. Ideals of commutative rings
You can also pass in a specific ideal type:

```python
sage: from sage.rings.ideal import Ideal_pid
sage: I = Ideal(x^2+3,ideal_class=Ideal_pid)
sage: type(I)
<class 'sage.rings.ideal.Ideal_pid'>
```

```python
class sage.rings.ideal.Ideal_fractional(ring, gens, coerce=True)
    Bases: Ideal_generic
    Fractional ideal of a ring.
    See Ideal().

class sage.rings.ideal.Ideal_generic(ring, gens, coerce=True)
    Bases: MonoidElement
    An ideal.
    See Ideal().

    absolute_norm()
    Returns the absolute norm of this ideal.
    In the general case, this is just the ideal itself, since the ring it lies in can’t be implicitly assumed to be an extension of anything.
    We include this function for compatibility with cases such as ideals in number fields.

    Todo: Implement this method.
```

**EXAMPLES:**

```python
sage: R.<t> = GF(9, names='a')[]  # needs sage.rings.finite_rings
sage: I = R.ideal(t^4 + t + 1)  # needs sage.rings.finite_rings
sage: I.absolute_norm()  # needs sage.rings.finite_rings
Traceback (most recent call last):
  ...
NotImplementedError
```

**apply_morphism(phi)**

Apply the morphism phi to every element of this ideal. Returns an ideal in the domain of phi.

**EXAMPLES:**

```python
sage: # needs sage.rings.real_mpfr
sage: psi = CC['x'].hom([-CC['x'].0])
sage: J = ideal([CC['x'].0 + 1]); J
Principal ideal (x + 1.00000000000000) of Univariate Polynomial Ring in x over Complex Field with 53 bits of precision
sage: psi(J)
Principal ideal (x - 1.00000000000000) of Univariate Polynomial Ring in x over Complex Field with 53 bits of precision
```

(continues on next page)
sage: J.apply_morphism(psi)
Principal ideal (x - 1.00000000000000) of Univariate Polynomial Ring in x
over Complex Field with 53 bits of precision

sage: psi = ZZ['x'].hom([-ZZ['x'].0])
sage: J = ideal([ZZ['x'].0, 2]); J
Ideal (x, 2) of Univariate Polynomial Ring in x over Integer Ring

sage: psi(J)
Ideal (-x, 2) of Univariate Polynomial Ring in x over Integer Ring

associated_primes()
Return the list of associated prime ideals of this ideal.

EXAMPLES:

sage: R = ZZ['x']
sage: I = R.ideal(7)
sage: I.associated_primes()
Traceback (most recent call last):
  ...
NotImplementedError

base_ring()
Returns the base ring of this ideal.

EXAMPLES:

sage: R = ZZ
sage: I = 3*R; I
Principal ideal (3) of Integer Ring
sage: J = 2*I; J
Principal ideal (6) of Integer Ring
sage: I.base_ring(); J.base_ring()
Integer Ring
Integer Ring

We construct an example of an ideal of a quotient ring:

sage: R = PolynomialRing(QQ, 'x'); x = R.gen()
sage: I = R.ideal(x^2 - 2)
sage: I.base_ring()
Rational Field

And $p$-adic numbers:

sage: R = Zp(7, prec=10); R
7-adic Ring with capped relative precision 10
sage: I = 7*R; I
Principal ideal (7 + O(7^11)) of 7-adic Ring with capped relative precision 10
sage: I.base_ring()  # needs sage.rings.padics
7-adic Ring with capped relative precision 10

category()

Return the category of this ideal.

**Note:** category is dependent on the ring of the ideal.

EXAMPLES:

```sage
sage: P.<x> = ZZ[]
sage: I = ZZ.ideal(7)
sage: J = P.ideal(7,x)
sage: K = P.ideal(7)
sage: I.category()
Category of ring ideals in Integer Ring
sage: J.category()
Category of ring ideals in Univariate Polynomial Ring in x over Integer Ring
sage: K.category()
Category of ring ideals in Univariate Polynomial Ring in x over Integer Ring
```

embedded_primes()

Return the list of embedded primes of this ideal.

EXAMPLES:

```sage
sage: R.<x, y> = QQ[]
sage: I = R.ideal(x^2, x*y)
sage: I.embedded_primes()  # needs sage.libs.singular
[Ideal (y, x) of Multivariate Polynomial Ring in x, y over Rational Field]
```

free_resolution(*args, **kwds)

Return a free resolution of self.

For input options, see `FreeResolution`.

EXAMPLES:

```sage
sage: R.<x> = PolynomialRing(QQ)
sage: I = R.ideal([x^4 + 3*x^2 + 2])
sage: I.free_resolution()  # needs sage.modules
S^1 <-- S^1 <-- 0
```

gen(i)

Return the i-th generator in the current basis of this ideal.

EXAMPLES:
sage: P.<x,y> = PolynomialRing(QQ,2)
sage: I = Ideal([x,y+1]); I
Ideal (x, y + 1) of Multivariate Polynomial Ring in x, y over Rational Field
sage: I.gen(1)
y + 1
sage: ZZ.ideal(5,10).gen()
5

gens()

Return a set of generators / a basis of self.
This is the set of generators provided during creation of this ideal.

EXAMPLES:

sage: P.<x,y> = PolynomialRing(QQ,2)
sage: I = Ideal([x,y+1]); I
Ideal (x, y + 1) of Multivariate Polynomial Ring in x, y over Rational Field
sage: I.gens()
[x, y + 1]
sage: ZZ.ideal(5,10).gens()
(5,)

gens_reduced()

Same as gens() for this ideal, since there is currently no special gens_reduced algorithm implemented for this ring.

This method is provided so that ideals in \( \mathbb{Z} \) have the method gens_reduced(), just like ideals of number fields.

EXAMPLES:

sage: ZZ.ideal(5).gens_reduced()
(5,)

graded_free_resolution(*args, **kwds)

Return a graded free resolution of self.

For input options, see GradedFiniteFreeResolution.

EXAMPLES:

sage: R.<x> = PolynomialRing(QQ)
sage: I = R.ideal([x^3])
sage: I.graded_free_resolution()  # needs sage.modules
S(0) <-- S(-3) <-- 0

is_maximal()

Return True if the ideal is maximal in the ring containing the ideal.

Todo: This is not implemented for many rings. Implement it!
EXAMPLES:

```python
sage: R = ZZ
sage: I = R.ideal(7)
sage: I.is_maximal()
True
sage: R.ideal(16).is_maximal()
False
sage: S = Integers(8)
sage: S.ideal(0).is_maximal()
False
sage: S.ideal(2).is_maximal()
True
sage: S.ideal(4).is_maximal()
False
```

**is_primary** *(P=None)*

Returns True if this ideal is primary (or \(P\)-primary, if a prime ideal \(P\) is specified).

Recall that an ideal \(I\) is primary if and only if \(I\) has a unique associated prime (see page 52 in [AM1969]). If this prime is \(P\), then \(I\) is said to be \(P\)-primary.

**INPUT:**

* \(P\) - (default: None) a prime ideal in the same ring

**EXAMPLES:**

```python
sage: R.<x, y> = QQ[]
sage: I = R.ideal([x^2, x*y])
sage: I.is_primary()
# → needs sage.libs.singular
False
sage: J = I.primary_decomposition()[1]; J
Ideal (y, x^2) of Multivariate Polynomial Ring in x, y over Rational Field
sage: J.is_primary()
# → needs sage.libs.singular
True
sage: J.is_prime()
# → needs sage.libs.singular
False
```

Some examples from the Macaulay2 documentation:

```python
sage: # needs sage.rings.finite_rings
sage: R.<x, y, z> = GF(101)[]
sage: I = R.ideal([y^6])
sage: I.is_primary()
# → needs sage.libs.singular
True
sage: I.is_primary(R.ideal([y]))
# → needs sage.libs.singular
True
sage: I = R.ideal([x^4, y^7])
sage: I.is_primary()
# (continues on next page)
```

(continues on next page)
needs sage.libs.singular

True

sage: I = R.ideal([x*y, y^2])

sage: I.is_primary()  # needs sage.libs.singular
False

Note: This uses the list of associated primes.

is_prime()

Return True if this ideal is prime.

EXAMPLES:

sage: R.<x, y> = QQ[]
sage: I = R.ideal([x, y])
sage: I.is_prime()  # a maximal ideal
# needs sage.libs.singular
True

sage: I = R.ideal([x^2 - y])
sage: I.is_prime()  # a non-maximal prime ideal
# needs sage.libs.singular
True

sage: I = R.ideal([x^2, y])
sage: I.is_prime()  # a non-prime primary ideal
# needs sage.libs.singular
False

sage: I = R.ideal([x^2, x*y])
sage: I.is_prime()  # a non-prime non-primary ideal
# needs sage.libs.singular
False

sage: S = Integers(8)
sage: S.ideal(0).is_prime()
False

sage: S.ideal(2).is_prime()
True

sage: S.ideal(4).is_prime()
False

Note that this method is not implemented for all rings where it could be:

sage: R.<x> = ZZ[]
sage: I = R.ideal(7)
sage: I.is_prime()  # when implemented, should be True
Traceback (most recent call last):
...    Not ImplementedError

Note: For general rings, uses the list of associated primes.
**is_principal()**

Returns True if the ideal is principal in the ring containing the ideal.

**Todo:** Code is naive. Only keeps track of ideal generators as set during initialization of the ideal. (Can the base ring change? See example below.)

**EXAMPLES:**
```python
sage: R.<x> = ZZ[]
sage: I = R.ideal(2, x)
sage: I.is_principal()
Traceback (most recent call last):
... 
NotImplementedError
sage: J = R.base_extend(QQ).ideal(2, x)
sage: J.is_principal()
True
```

**is_trivial()**

Return True if this ideal is (0) or (1).

**minimal_associated_primes()**

Return the list of minimal associated prime ideals of this ideal.

**EXAMPLES:**
```python
sage: R = ZZ['x']
sage: I = R.ideal(7)
sage: I.minimal_associated_primes()
Traceback (most recent call last):
... 
NotImplementedError
```

**ngens()**

Return the number of generators in the basis.

**EXAMPLES:**
```python
sage: P.<x,y> = PolynomialRing(QQ,2)
sage: I = Ideal([x,y+1]); I
Ideal (x, y + 1) of Multivariate Polynomial Ring in x, y over Rational Field
sage: I.ngens()
2
sage: ZZ.ideal(5,10).ngens()
1
```

**norm()**

Returns the norm of this ideal.

In the general case, this is just the ideal itself, since the ring it lies in can't be implicitly assumed to be an extension of anything.

We include this function for compatibility with cases such as ideals in number fields.

**EXAMPLES:**
sage: R.<t> = GF(8, names='a')[]  # needs sage.rings.finite_rings
sage: I = R.ideal(t^4 + t + 1)  # needs sage.rings.finite_rings
sage: I.norm()                  # needs sage.rings.finite_rings
Principal ideal (t^4 + t + 1) of Univariate Polynomial Ring in t
over Finite Field in a of size 2^3

primary_decomposition()

Return a decomposition of this ideal into primary ideals.

EXAMPLES:

sage: R = ZZ['x']
sage: I = R.ideal(7)
sage: I.primary_decomposition()
Traceback (most recent call last):
... Not ImplementedError

random_element(*args, **kwds)

Return a random element in this ideal.

EXAMPLES:

sage: P.<a,b,c> = GF(5)[[]]
sage: I = P.ideal([a^2, a*b + c, c^3])
sage: I.random_element()  # random
2*a^5*c + a^2*b*c^4 + ... + O(a, b, c)^13

reduce(f)

Return the reduction of the element of $f$ modulo self.

This is an element of $R$ that is equivalent modulo $I$ to $f$ where $I$ is self.

EXAMPLES:

sage: ZZ.ideal(5).reduce(17)
2
sage: parent(ZZ.ideal(5).reduce(17))
Integer Ring

ring()

Return the ring containing this ideal.

EXAMPLES:

sage: R = ZZ
sage: I = 3*R; I
Principal ideal (3) of Integer Ring
sage: J = 2*I; J
Principal ideal (6) of Integer Ring
sage: I.ring(); J.ring()
Integer Ring
Integer Ring

2.1. Ideals of commutative rings
Note that `self.ring()` is different from `self.base_ring()`

```
sage: R = PolynomialRing(QQ, 'x'); x = R.gen()
sage: I = R.ideal(x^2 - 2)
sage: I.base_ring()
Rational Field
sage: I.ring()
Univariate Polynomial Ring in x over Rational Field
```

Another example using polynomial rings:

```
sage: R = PolynomialRing(QQ, 'x'); x = R.gen()
sage: I = R.ideal(x^2 - 3)
sage: I.ring()
Univariate Polynomial Ring in x over Rational Field

sage: Rbar = R.quotient(I, names='a')
    # needs sage.libs.pari
sage: S = PolynomialRing(Rbar, 'y'); y = Rbar.gen(); S
    # needs sage.libs.pari
Univariate Polynomial Ring in y over
Univariate Quotient Polynomial Ring in a over Rational Field with modulus x^2 - 3

sage: J = S.ideal(y^2 + 1)
    # needs sage.libs.pari
sage: J.ring()
    # needs sage.libs.pari
Univariate Polynomial Ring in y over
Univariate Quotient Polynomial Ring in a over Rational Field with modulus x^2 - 3
```

### class `sage.rings.ideal.Ideal_pid(ring, gen)`

Bases: `Ideal_principal`

An ideal of a principal ideal domain.

See `Ideal()`.

**gcd**(other)

Returns the greatest common divisor of the principal ideal with the ideal other; that is, the largest principal ideal contained in both the ideal and other

**Todo:** This is not implemented in the case when other is neither principal nor when the generator of self is contained in other. Also, it seems that this class is used only in PIDs—is this redundant?

**Note:** The second example is broken.

### EXAMPLES:

An example in the principal ideal domain $\mathbb{Z}$:

```
sage: R = ZZ
sage: I = R.ideal(42)
sage: J = R.ideal(70)
```
sage: I.gcd(J)
Principal ideal (14) of Integer Ring
sage: J.gcd(I)
Principal ideal (14) of Integer Ring

**is_maximal()**

Returns whether this ideal is maximal.

Principal ideal domains have Krull dimension 1 (or 0), so an ideal is maximal if and only if it’s prime (and nonzero if the ring is not a field).

**EXAMPLES:**

```sage
sage: # needs sage.rings.finite_rings
sage: R.<t> = GF(5)[]
sage: p = R.ideal(t^2 + 2)
sage: p.is_maximal()
True
sage: p = R.ideal(t^2 + 1)
sage: p.is_maximal()
False
sage: p = R.ideal(0)
```

```sage
sage: p.is_maximal()
False
sage: p = R.ideal(1)
```

```sage
sage: p.is_maximal()
False
```

**is_prime()**

Return True if the ideal is prime.

This relies on the ring elements having a method `is_irreducible()` implemented, since an ideal \((a)\) is prime iff \(a\) is irreducible (or 0).

**EXAMPLES:**

```sage
sage: ZZ.ideal(2).is_prime()
True
sage: ZZ.ideal(-2).is_prime()
True
sage: ZZ.ideal(4).is_prime()
False
sage: ZZ.ideal(0).is_prime()
True
```

```sage
sage: R.<x> = QQ[]
sage: P = R.ideal(x^2 + 1); P
Principal ideal (x^2 + 1) of Univariate Polynomial Ring in x over Rational Field
sage: P.is_prime()
# needs sage.libs.pari
```

In fields, only the zero ideal is prime:
General Rings, Ideals, and Morphisms, Release 10.2

\begin{verbatim}
sage: RR.ideal(0).is_prime()
True
sage: RR.ideal(7).is_prime()
False
\end{verbatim}

\texttt{reduce}(f)

Return the reduction of \( f \) modulo \texttt{self}.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: I = 8*ZZ
sage: I.reduce(10)
2
sage: n = 10; n.mod(I)
2
\end{verbatim}

\texttt{residue_field}()

Return the residue class field of this ideal, which must be prime.

\textbf{Todo:} Implement this for more general rings. Currently only defined for \texttt{ZZ} and for number field orders.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: # needs sage.libs.pari
sage: P = ZZ.ideal(61); P
Principal ideal (61) of Integer Ring
sage: F = P.residue_field(); F
Residue field of Integers modulo 61
sage: pi = F.reduction_map(); pi
Partially defined reduction map:
   From: Rational Field
   To:   Residue field of Integers modulo 61
sage: pi(123/234)
6
sage: pi(1/61)
Traceback (most recent call last):
  ...
ZeroDivisionError: Cannot reduce rational 1/61 modulo 61: it has negative...

sage: lift = F.lift_map(); lift
Lifting map:
   From: Residue field of Integers modulo 61
   To:   Integer Ring
sage: lift(F(12345/67890))
33
sage: (12345/67890) % 61
33
\end{verbatim}

\textbf{class} \texttt{sage.rings.ideal.Ideal_principal}(\texttt{ring}, \texttt{gens}, \texttt{coerce=True})

\textbf{Bases:} \texttt{Ideal_generic}

A principal ideal.

See \texttt{Ideal()}. 

**divides** *(other)*

Return True if self divides other.

**EXAMPLES:**

```python
sage: P.<x> = PolynomialRing(QQ)
sage: I = P.ideal(x)
sage: J = P.ideal(x^2)
sage: I.divides(J)
True
sage: J.divides(I)
False
```

**gen** *(i=0)*

Return the generator of the principal ideal.

The generator is an element of the ring containing the ideal.

**EXAMPLES:**

A simple example in the integers:

```python
sage: R = ZZ
sage: I = R.ideal(7)
sage: J = R.ideal(7, 14)
sage: I.gen(); J.gen()
7
7
```

Note that the generator belongs to the ring from which the ideal was initialized:

```python
sage: R.<x> = ZZ[]
sage: I = R.ideal(x)
sage: J = R.base_extend(QQ).ideal(2,x)
sage: a = I.gen(); a
x
sage: b = J.gen(); b
1
sage: a.base_ring()
Integer Ring
sage: b.base_ring()
Rational Field
```

**is_principal**

Returns True if the ideal is principal in the ring containing the ideal. When the ideal construction is explicitly principal (i.e. when we define an ideal with one element) this is always the case.

**EXAMPLES:**

Note that Sage automatically coerces ideals into principal ideals during initialization:

```python
sage: R.<x> = ZZ[]
sage: I = R.ideal(x)
sage: J = R.ideal(2,x)
sage: K = R.base_extend(QQ).ideal(2,x)
sage: I
```

(continues on next page)
Principal ideal (x) of Univariate Polynomial Ring in x over Integer Ring

```
sage: J
Ideal (2, x) of Univariate Polynomial Ring in x over Integer Ring
```

Principal ideal (1) of Univariate Polynomial Ring in x over Rational Field

```
sage: I.is_principal()
True
sage: K.is_principal()
True
```

```
sage.rings.ideal.Katsura(R, n=None, homog=False, singular=None)
n-th katsura ideal of R if R is coercible to Singular.

INPUT:

- R – base ring to construct ideal for
- n – (default: None) which katsura ideal of R. If None, then n is set to R.ngens().
- homog – if True a homogeneous ideal is returned using the last variable in the ideal (default: False)
- singular – singular instance to use

EXAMPLES:

```
sage: P.<x,y,z> = PolynomialRing(QQ, 3)
sage: I = sage.rings.ideal.Katsura(P, 3); I
Ideal (x + 2*y + 2*z - 1, x^2 + 2*y^2 + 2*z^2 - x, 2*x*y + 2*y*z - y)
of Multivariate Polynomial Ring in x, y, z over Rational Field
```

```
sage: Q.<x> = PolynomialRing(QQ, implementation="singular")
sage: J = sage.rings.ideal.Katsura(Q,1); J
Ideal (x - 1) of Multivariate Polynomial Ring in x over Rational Field
```

```
sage.rings.ideal.is_Ideal(x)
Return True if object is an ideal of a ring.

EXAMPLES:

A simple example involving the ring of integers. Note that Sage does not interpret rings objects themselves as ideals. However, one can still explicitly construct these ideals:

```
sage: from sage.rings.ideal import is_Ideal
sage: R = ZZ
sage: is_Ideal(R)
False
sage: is_Ideal(1*R)
Principal ideal (1) of Integer Ring
True
sage: is_Ideal(0*R)
```
Principal ideal (0) of Integer Ring
True

Sage recognizes ideals of polynomial rings as well:

```python
sage: R = PolynomialRing(QQ, 'x'); x = R.gen()
sage: I = R.ideal(x^2 + 1); I
Principal ideal (x^2 + 1) of Univariate Polynomial Ring in x over Rational Field
sage: is_Ideal(I)
True
sage: is_Ideal((x^2 + 1)*R)
True
```

### 2.2 Monoid of ideals in a commutative ring

**WARNING:** This is used by some rings that are not commutative!

```python
sage: MS = MatrixSpace(QQ, 3, 3)

 returns sage.modules
sage: type(MS.ideal(MS.one()).parent())
```

sage.rings.ideal_monoid.IdealMonoid(R)

Return the monoid of ideals in the ring R.

**EXAMPLES:**

```python
sage: R = QQ['x']
sage: from sage.rings.ideal_monoid import IdealMonoid
sage: IdealMonoid(R)
Monoid of ideals of Univariate Polynomial Ring in x over Rational Field
```

```python
class sage.rings.ideal_monoid.IdealMonoid_c(R)
    Bases: Parent
    The monoid of ideals in a commutative ring.
    Element
        alias of Ideal_generic
    ring()
        Return the ring of which this is the ideal monoid.
        EXAMPLES:
```

```python
sage: R = QuadraticField(-23, 'a')
from sage.rings.ideal_monoid import IdealMonoid
M = IdealMonoid(R); M.ring() is R
```

2.2. Monoid of ideals in a commutative ring 51
2.3 Ideals of non-commutative rings

Generic implementation of one- and two-sided ideals of non-commutative rings.

AUTHOR:

• Simon King (2011-03-21), <simon.king@uni-jena.de>, github issue #7797.

EXAMPLES:

```
sage: MS = MatrixSpace(ZZ,2,2)
sage: MS*MS([[0,1,-2,3]])
Left Ideal
(  [0 1]
   [-2 3])
of Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
sage: MS([[0,1,-2,3]])*MS
Right Ideal
(  [0 1]
   [-2 3])
of Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
sage: MS*MS([[0,1,-2,3]])*MS
Twosided Ideal
(  [0 1]
   [-2 3])
of Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
```

See letterplace_ideal for a more elaborate implementation in the special case of ideals in free algebras.

```
class sage.rings.noncommutative_ideals.IdealMonoid_nc(R)
    Bases: IdealMonoid_c

    Base class for the monoid of ideals over a non-commutative ring.

    Note: This class is essentially the same as IdealMonoid_c, but does not complain about non-commutative rings.

    EXAMPLES:

    sage: MS = MatrixSpace(ZZ,2,2)
sage: MS.ideal_monoid()
Monoid of ideals of Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
```

```
class sage.rings.noncommutative_ideals.Ideal_nc(ring, gens, coerce=True, side='twosided')
    Bases: Ideal_generic

    Generic non-commutative ideal.

    All fancy stuff such as the computation of Groebner bases must be implemented in sub-classes. See LetterplaceIdeal for an example.
```
EXAMPLES:

```
sage: MS = MatrixSpace(QQ,2,2)
sage: I = MS*[MS.1,MS.2]; I
Left Ideal
  (
    [0 1]
    [0 0],
    [0 0]
    [1 0]
  )
of Full MatrixSpace of 2 by 2 dense matrices over Rational Field
sage: [MS.1,MS.2]*MS
Right Ideal
  (
    [0 1]
    [0 0],
    [0 0]
    [1 0]
  )
of Full MatrixSpace of 2 by 2 dense matrices over Rational Field
sage: MS*[MS.1,MS.2]*MS
Twosided Ideal
  (
    [0 1]
    [0 0],
    [0 0]
    [1 0]
  )
of Full MatrixSpace of 2 by 2 dense matrices over Rational Field
```

`side()`

Return a string that describes the sidedness of this ideal.

EXAMPLES:

```
sage: # needs sage.combinat
sage: A = SteenrodAlgebra(2)
sage: IL = A*[A.1+A.2,A.1^2]
sage: IR = [A.1+A.2,A.1^2]*A
sage: IT = A*[A.1+A.2,A.1^2]*A
sage: IL.side()
'left'
sage: IR.side()
'right'
sage: IT.side()
'twosided'
```
3.1 Homomorphisms of rings

We give a large number of examples of ring homomorphisms.

EXAMPLES:

Natural inclusion \( \mathbb{Z} \rightarrow \mathbb{Q} \):

```sage
sage: H = Hom(ZZ, QQ)
sage: phi = H([1])
sage: phi(10)
10
sage: phi(3/1)
3
sage: phi(2/3)
Traceback (most recent call last):
  ...TypeError: 2/3 fails to convert into the map's domain Integer Ring, but a `pushforward` method is not properly implemented
```

There is no homomorphism in the other direction:

```sage
sage: H = Hom(QQ, ZZ)
sage: H([1])
Traceback (most recent call last):
  ...ValueError: relations do not all (canonically) map to 0 under map determined by images of generators
```

EXAMPLES:

Reduction to finite field:

```sage
# needs sage.rings.finite_rings
sage: H = Hom(ZZ, GF(9, 'a'))
sage: phi = H([1])
sage: phi(5)
2
sage: psi = H([4])
sage: psi(5)
2
```
Map from single variable polynomial ring:

```
sage: R.<x> = ZZ[]
sage: phi = R.hom([2], GF(5)); phi
Ring morphism:
  From: Univariate Polynomial Ring in x over Integer Ring
  To:   Finite Field of size 5
  Defn: x |--> 2
sage: phi(x + 12)
4
```

Identity map on the real numbers:

```
sage: # needs sage.rings.real_mpfr
sage: f = RR.hom([RR(1)]); f
Ring endomorphism of Real Field with 53 bits of precision
  Defn: 1.00000000000000 |--> 1.00000000000000
sage: f(2.5)
2.50000000000000
sage: f = RR.hom([2.0])
Traceback (most recent call last):
... ValueError: relations do not all (canonically) map to 0
under map determined by images of generators
```

Homomorphism from one precision of field to another.

From smaller to bigger doesn't make sense:

```
sage: R200 = RealField(200)   # needs sage.rings.real_mpfr
sage: f = RR.hom( R200 )     # needs sage.rings.real_mpfr
Traceback (most recent call last):
... TypeError: natural coercion morphism from Real Field with 53 bits of precision
to Real Field with 200 bits of precision not defined
```

From bigger to small does:

```
sage: f = RR.hom(RealField(15))   # needs sage.rings.real_mpfr
sage: f(2.5)   # needs sage.rings.real_mpfr
2.500
sage: f(RR.pi())   # needs sage.rings.real_mpfr
3.142
```

Inclusion map from the reals to the complexes:

```
sage: # needs sage.rings.real_mpfr
sage: i = RR.hom([CC(1)]); i
Ring morphism:
  From: Real Field with 53 bits of precision
```

(continues on next page)
To: Complex Field with 53 bits of precision
Defn: 1.00000000000000 |--> 1.00000000000000

```
sage: i(RR('3.1'))
3.10000000000000
```

A map from a multivariate polynomial ring to itself:

```
sage: R.<x,y,z> = PolynomialRing(QQ, 3)
sage: phi = R.hom([y, z, x^2]); phi
Ring endomorphism of Multivariate Polynomial Ring in x, y, z over Rational Field
Defn: x |--> y
       y |--> z
       z |--> x^2

sage: phi(x + y + z)
x^2 + y + z
```

An endomorphism of a quotient of a multi-variate polynomial ring:

```
sage: # needs sage.libs.singular
sage: R.<x,y> = PolynomialRing(QQ)
sage: S.<a,b> = quo(R, ideal(1 + y^2))
sage: phi = S.hom([a^2, -b]); phi
Ring endomorphism of Quotient of Multivariate Polynomial Ring in x, y
   over Rational Field by the ideal (y^2 + 1)
Defn: a |--> a^2
       b |--> -b

sage: phi(b)
-b
sage: phi(a^2 + b^2)
a^4 - 1
```

The reduction map from the integers to the integers modulo 8, viewed as a quotient ring:

```
sage: R = ZZ.quo(8*ZZ)
sage: pi = R.cover(); pi
Ring morphism:
   From: Integer Ring
   To: Ring of integers modulo 8
   Defn: Natural quotient map

sage: pi.domain()
Integer Ring
sage: pi.codomain()
Ring of integers modulo 8

sage: pi(10)
2
sage: pi.lift()
Set-theoretic ring morphism:
   From: Ring of integers modulo 8
   To: Integer Ring
   Defn: Choice of lifting map

sage: pi.lift(13)
5
```

3.1. Homomorphisms of rings
Inclusion of \( \text{GF}(2) \) into \( \text{GF}(4, 'a') \):

```python
sage: # needs sage.rings.finite_rings
sage: k = GF(2)
```

```python
def i = k.hom(GF(4, 'a'))
sage: i
Ring morphism:
   From: Finite Field of size 2
   To:   Finite Field in a of size 2^2
   Defn: 1 |--> 1
sage: i(0)
0
sage: a = i(1); a.parent()
Finite Field in a of size 2^2
```

We next compose the inclusion with reduction from the integers to \( \text{GF}(2) \):

```python
sage: # needs sage.rings.finite_rings
sage: pi = ZZ.hom(k); pi
Natural morphism:
   From: Integer Ring
   To:   Finite Field of size 2
sage: f = i * pi; f
Composite map:
   From: Integer Ring
   To:   Finite Field in a of size 2^2
   Defn: Natural morphism:
      From: Integer Ring
      To:   Finite Field of size 2
      then
      Ring morphism:
         From: Finite Field of size 2
         To:   Finite Field in a of size 2^2
         Defn: 1 |--> 1
sage: a = f(5); a
1
sage: a.parent()
Finite Field in a of size 2^2
```

Inclusion from \( \mathbb{Q} \) to the 3-adic field:

```python
sage: # needs sage.rings.padics
sage: phi = QQ.hom(Qp(3, print_mode='series'))
sage: phi
Ring morphism:
   From: Rational Field
   To: 3-adic Field with capped relative precision 20
sage: phi.codomain()
3-adic Field with capped relative precision 20
sage: phi(394)
1 + 2*3 + 3^2 + 2*3^3 + 3^4 + 3^5 + O(3^20)
```

An automorphism of a quotient of a univariate polynomial ring:
sage: # needs sage.libs.pari
sage: R.<x> = PolynomialRing(QQ)
sage: S.<sqrt2> = R.quo(x^2 - 2)
sage: sqrt2^2
2
sage: (3+sqrt2)^10
993054*sqrt2 + 1404491
sage: c = S.hom([-sqrt2])
sage: c(1+sqrt2)
-sqrt2 + 1

Note that Sage verifies that the morphism is valid:

sage: (1 - sqrt2)^2
˓→ needs sage.libs.pari
-2*sqrt2 + 3
sage: c = S.hom([1 - sqrt2])  # this is not valid ˓→ needs sage.libs.pari
Traceback (most recent call last):
...
ValueError: relations do not all (canonically) map to 0
under map determined by images of generators

Endomorphism of power series ring:

sage: R.<t> = PowerSeriesRing(QQ, default_prec=10); R
Power Series Ring in t over Rational Field
sage: f = R.hom([t^2]); f
Ring endomorphism of Power Series Ring in t over Rational Field
   Defn: t |--> t^2
sage: s = 1/(1 + t); s
1 - t + t^2 - t^3 + t^4 - t^5 + t^6 - t^7 + t^8 - t^9 + O(t^10)
sage: f(s)
1 - t^2 + t^4 - t^6 + t^8 - t^10 + t^12 - t^14 + t^16 - t^18 + O(t^20)

Frobenius on a power series ring over a finite field:

sage: R.<t> = PowerSeriesRing(GF(5))
sage: f = R.hom([t^5]); f
Ring endomorphism of Power Series Ring in t over Finite Field of size 5
   Defn: t |--> t^5
sage: a = 2 + t + 3*t^2 + 4*t^3 + O(t^4)
sage: b = 1 + t + 2*t^2 + t^3 + O(t^5)
sage: f(a)
2 + t^5 + 3*t^10 + 4*t^15 + O(t^20)
sage: f(b)
1 + t^5 + 2*t^10 + t^15 + O(t^25)
sage: f(a*b)
2 + 3*t^5 + 3*t^10 + t^15 + O(t^20)
sage: f(a)^2*f(b)
2 + 3*t^5 + 3*t^10 + t^15 + O(t^20)

Homomorphism of Laurent series ring:

3.1. Homomorphisms of rings
sage: R.<t> = LaurentSeriesRing(QQ, 10)
sage: f = R.hom([t^3 + t]); f
Ring endomorphism of Laurent Series Ring in t over Rational Field
   Defn: t |---> t + t^3
sage: s = 2/t^2 + 1/(1 + t); s
2*t^-2 + 1 - t + t^2 - t^3 + t^4 - t^5 + t^6 - t^7 + t^8 - t^9 + O(t^10)
sage: f(s)
2*t^-2 - 3 - t + 7*t^2 - 2*t^3 - 5*t^4 - 4*t^5 + 16*t^6 - 9*t^7 + O(t^8)
sage: f = R.hom([t^3]); f
Ring endomorphism of Laurent Series Ring in t over Rational Field
   Defn: t |---> t^3
sage: f(s)
2*t^-6 + 1 - t^3 + t^6 - t^9 + t^12 - t^15 + t^18 - t^21 + t^24 - t^27 + O(t^30)

Note that the homomorphism must result in a converging Laurent series, so the valuation of the image of the generator must be positive:

sage: R.hom([1/t])
Traceback (most recent call last):
... ValueError: relations do not all (canonically) map to 0 under map determined by images of generators
sage: R.hom([1])
Traceback (most recent call last):
... ValueError: relations do not all (canonically) map to 0 under map determined by images of generators

Complex conjugation on cyclotomic fields:

sage: # needs sage.rings.number_field
sage: K.<zeta7> = CyclotomicField(7)
sage: c = K.hom([1/zeta7]); c
Ring endomorphism of Cyclotomic Field of order 7 and degree 6
   Defn: zeta7 |--> -zeta7^5 - zeta7^4 - zeta7^3 - zeta7^2 - zeta7 - 1
sage: a = (1+zeta7)^5; a
zeta7^5 + 5*zeta7^4 + 10*zeta7^3 + 10*zeta7^2 + 5*zeta7 + 1
sage: c(a)
-5*zeta7^5 - 5*zeta7^4 - 4*zeta7^2 - 5*zeta7 - 4
sage: c(zeta7 + 1/zeta7) # this element is obviously fixed by inversion
-zeta7^5 - zeta7^4 - zeta7^3 - zeta7^2 - 1
sage: zeta7 + 1/zeta7
-zeta7^5 - zeta7^4 - zeta7^3 - zeta7^2 - 1

Embedding a number field into the reals:

sage: # needs sage.rings.number_field
sage: R.<x> = PolynomialRing(QQ)
sage: K.<beta> = NumberField(x^3 - 2)
sage: alpha = RR(2)^(1/3); alpha
1.25992104989487
sage: i = K.hom([alpha],check=False); i
Ring morphism:
From: Number Field in beta with defining polynomial $x^3 - 2$
To: Real Field with 53 bits of precision
Defn: beta |--> 1.25992104989487

\begin{verbatim}
sage: i(beta)
1.25992104989487
sage: i(beta^3)
2.00000000000000
sage: i(beta^2 + 1)
2.58740105196820
\end{verbatim}

An example from Jim Carlson:

\begin{verbatim}
sage: K = QQ  # by the way :-)
sage: R.<a,b,c,d> = K[]; R
Multivariate Polynomial Ring in a, b, c, d over Rational Field
sage: S.<u> = K[]; S
Univariate Polynomial Ring in u over Rational Field
sage: f = R.hom([0,0,0,u], S); f
Ring morphism:
From: Multivariate Polynomial Ring in a, b, c, d over Rational Field
To:   Univariate Polynomial Ring in u over Rational Field
Defn: a |--> 0
       b |--> 0
       c |--> 0
       d |--> u
sage: f(a + b + c + d)
u
sage: f((a+b+c+d)^2)
u^2
\end{verbatim}

class sage.rings.morphism.FrobeniusEndomorphism_generic
Bases: RingHomomorphism

A class implementing Frobenius endomorphisms on rings of prime characteristic.

\texttt{power()}

Return an integer \( n \) such that this endomorphism is the \( n \)-th power of the absolute (arithmetic) Frobenius.

EXAMPLES:

\begin{verbatim}
sage: # needs sage.rings.finite_rings
sage: K.<u> = PowerSeriesRing(GF(5))
sage: Frob = K.frobenius_endomorphism()
sage: Frob.power()
1
sage: (Frob^9).power()
9
\end{verbatim}

class sage.rings.morphism.RingHomomorphism
Bases: RingMap

Homomorphism of rings.

\texttt{inverse()}

Return the inverse of this ring homomorphism if it exists.

\section{Homomorphisms of rings}
Raises a `ZeroDivisionError` if the inverse does not exist.

ALGORITHM:

By default, this computes a Gröbner basis of the ideal corresponding to the graph of the ring homomorphism.

EXAMPLES:

```sage
sage: R.<t> = QQ[]
sage: f = R.hom([2*t - 1], R)
sage: f.inverse()  # needs sage.libs.singular
Ring endomorphism of Univariate Polynomial Ring in t over Rational Field
  Defn: t |--> 1/2*t + 1/2
```

The following non-linear homomorphism is not invertible, but it induces an isomorphism on a quotient ring:

```sage
sage: # needs sage.libs.singular
sage: R.<x,y,z> = QQ[]
sage: f = R.hom([y*z, x*z, x*y], R)
sage: f.inverse()  # needs sage.libs.singular
Traceback (most recent call last):
  ...  
ZeroDivisionError: ring homomorphism not surjective
sage: f.is_injective()
True
sage: Q.<x,y,z> = R.quotient(x*y*z - 1)
sage: g = Q.hom([y*z, x*z, x*y], Q)
sage: g.inverse()
Ring endomorphism of Quotient of Multivariate Polynomial Ring in x, y, z over Rational Field by the ideal (x*y*z - 1)
  Defn: x |--> y*z
  y |--> x*z
  z |--> x*y
```

Homomorphisms over the integers are supported:

```sage
sage: S.<x,y> = ZZ[]
sage: f = S.hom([x + 2*y, x + 3*y], S)
sage: f.inverse()  # needs sage.libs.singular
Ring endomorphism of Multivariate Polynomial Ring in x, y over Integer Ring
  Defn: x |--> 3*x - 2*y
  y |--> -x + y
sage: (f.inverse() * f).is_identity()  # needs sage.libs.singular
True
```

The following homomorphism is invertible over the rationals, but not over the integers:

```sage
sage: g = S.hom([x + y, x - y - 2], S)
sage: g.inverse()  # needs sage.libs.singular
```

(continues on next page)
This example by M. Nagata is a wild automorphism:

```python
sage: R.<x,y,z> = QQ[]
sage: sigma = R.hom([x - 2*y*(z*x+y^2) - z*(z*x+y^2)^2, 
    y + z*(z*x+y^2), z], R)
```

We compute the triangular automorphism that converts moments to cumulants, as well as its inverse, using the moment generating function. The choice of a term ordering can have a great impact on the computation time of a Gröbner basis, so here we choose a weighted ordering such that the images of the generators are homogeneous polynomials.

```python
sage: d = 12
sage: T = TermOrder('wdegrevlex', [1..d])
sage: R = PolynomialRing(QQ, ['x%s' % j for j in (1..d)], order=T)
sage: S.<t> = PowerSeriesRing(R)
sage: egf = S([0] + list(R.gens())).ogf_to_egf().exp(prec=d+1)
sage: phi = R.hom(egf.egf_to_ogf().list()[1:], R)
sage: phi.im_gens()[:5]
[x1, 
 x1^2 + x2, 
 x1^3 + 3*x1*x2 + x3, 
 x1^4 + 6*x1^2*x2 + 3*x2^2 + 4*x1*x3 + x4, 
 x1^5 + 10*x1^3*x2 + 15*x1*x2^2 + 10*x1^2*x3 + 10*x2*x3 + 5*x1*x4 + x5]
sage: all(p.is_homogeneous() for p in phi.im_gens()) # needs sage.libs.singular
True
sage: phi.inverse().im_gens()[:5] # needs sage.libs.singular
[x1, 
 -x1^2 + x2, 
 2*x1^3 - 3*x1*x2 + x3, 
 -6*x1*x4 + 12*x1^2*x2 - 3*x2^2 - 4*x1*x3 + x4, 
 24*x1^5 - 60*x1^3*x2 + 30*x1*x2^2 + 20*x1^2*x3 - 10*x2*x3 - 5*x1*x4 + x5]
```

(continues on next page)
General Rings, Ideals, and Morphisms, Release 10.2

sage: (phi.inverse() * phi).is_identity()  
needs sage.libs.singular
True

Automorphisms of number fields as well as Galois fields are supported:

sage: K.<zeta7> = CyclotomicField(7)  
needs sage.rings.number_field
sage: c = K.hom([1/zeta7])  
needs sage.rings.number_field
sage: (c.inverse() * c).is_identity()  
needs sage.libs.singular sage.rings.number_field
True

sage: F.<t> = GF(7^3)  
needs sage.rings.finite_rings
sage: f = F.hom(t^7, F)  
needs sage.rings.finite_rings
sage: (f.inverse() * f).is_identity()  
needs sage.libs.singular sage.rings.finite_rings
True

An isomorphism between the algebraic torus and the circle over a number field:

sage: # needs sage.rings.number_field
sage: K.<i> = QuadraticField(-1)

sage: A.<z,w> = K['z,w'].quotient('z*w - 1')

sage: B.<x,y> = K['x,y'].quotient('x^2 + y^2 - 1')

sage: f = A.hom([x + i*y, x - i*y], B)

sage: g = f.inverse()

sage: g.morphism_from_cover().im_gens()
[1/2*z + 1/2*w, (-1/2*i)*z + (1/2*i)*w]

sage: all(g(f(z)) == z for z in A.gens())
True

inverse_image(I)

Return the inverse image of an ideal or an element in the codomain of this ring homomorphism.

INPUT:

• I – an ideal or element in the codomain

OUTPUT:

For an ideal I in the codomain, this returns the largest ideal in the domain whose image is contained in I.

Given an element b in the codomain, this returns an arbitrary element a in the domain such that self(a) = b if one such exists. The element a is unique if this ring homomorphism is injective.

EXAMPLES:

sage: R.<x,y,z> = QQ[]
sage: S.<u,v> = QQ[]
sage: f = R.hom([u*v^2, u^v, v^2], S)
sage: I = S.ideal([u^6, u^5*v, u^4*v^2, u^3*v^3])
Under the above homomorphism, there exists an inverse image for every element that only involves monomials of even degree:

```sage
sage: [f.inverse_image(p) for p in [u^2, u^4, u*v + u^3*v^3]]
 needs sage.libs.singular
[x, x^2, x*y*z + y]
sage: f.inverse_image(u*v^2)
 needs sage.libs.singular
Traceback (most recent call last):
... ValueError: element u*v^2 does not have preimage
```

The image of the inverse image ideal can be strictly smaller than the original ideal:

```sage
sage: S.<u,v> = QQ['u,v'].quotient('v^2 - 2')
sage: f = QuadraticField(2).hom([v], S)
sage: I = S.ideal(u + v)
sage: J = f.inverse_image(I)
sage: J.is_zero()
True
sage: f(J) < I
True
```

Fractional ideals are not yet fully supported:

```sage
sage: K.<a> = NumberField(QQ['x'](x^2+2))
sage: f = K.hom([-a], K)
sage: I = K.ideal([a + 1])
sage: f.inverse_image(I)
 needs sage.libs.singular
Traceback (most recent call last):
... NotImplementedError: inverse image not implemented...
sage: f.inverse_image(K.ideal(0)).is_zero()
 needs sage.libs.singular
True
sage: f.inverse()(I)
 needs sage.rings.padics
Fractional ideal (-a + 1)
```

**ALGORITHM:**

By default, this computes a Gröbner basis of an ideal related to the graph of the ring homomorphism.
REFERENCES:

• Proposition 2.5.12 [DS2009]

**is_invertible()**

Return whether this ring homomorphism is bijective.

**EXAMPLES:**

```
sage: R.<x,y,z> = QQ[]
sage: R.hom([y*z, x*z, x*y], R).is_invertible() # needs sage.libs.singular
False
```

```
sage: Q.<x,y,z> = R.quotient(x*y*z - 1) # needs sage.libs.singular
sage: Q.hom([y*z, x*z, x*y], Q).is_invertible() # needs sage.libs.singular
True
```

**ALGORITHM:**

By default, this requires the computation of a Gröbner basis.

**is_surjective()**

Return whether this ring homomorphism is surjective.

**EXAMPLES:**

```
sage: R.<x,y,z> = QQ[]
sage: R.hom([y*z, x*z, x*y], R).is_surjective() # needs sage.libs.singular
False
```

```
sage: Q.<x,y,z> = R.quotient(x*y*z - 1) # needs sage.libs.singular
sage: Q.hom([y*z, x*z, x*y], Q).is_surjective() # needs sage.libs.singular
True
```

**ALGORITHM:**

By default, this requires the computation of a Gröbner basis.

**kernel()**

Return the kernel ideal of this ring homomorphism.

**EXAMPLES:**

```
sage: A.<x,y> = QQ[]
sage: B.<t> = QQ[]
sage: f = A.hom([t^4, t^3 - t^2], B)
sage: f.kernel() # needs sage.libs.singular
Ideal (y^4 - x^3 + 4*x^2*y - 2*x*y^2 + x^2)
of Multivariate Polynomial Ring in x, y over Rational Field
```

We express a Veronese subring of a polynomial ring as a quotient ring:
```python
sage: A.<a,b,c,d> = QQ[]
sage: B.<u,v> = QQ[]
sage: f = A.hom([u^3, u^2*v, u*v^2, v^3], B)
sage: f.kernel() == A.ideal(matrix.hankel([a, b, c], [d]).minors(2))  # needs sage.libs.singular
True
sage: Q = A.quotient(f.kernel())  # needs sage.libs.singular
sage: Q.hom(f.im_gens(), B).is_injective()  # needs sage.libs.singular
True
```

The Steiner-Roman surface:

```python
sage: R.<x,y,z> = QQ[]
sage: S = R.quotient(x^2 + y^2 + z^2 - 1)
sage: f = R.hom([x*y, x*z, y*z], S)  # needs sage.libs.singular
sage: f.kernel()  # needs sage.libs.singular
Ideal (x^2*y^2 + x^2*z^2 + y^2*z^2 - x*y*z)
of Multivariate Polynomial Ring in x, y, z over Rational Field
```

**lift**(x=None)

Return a lifting map associated to this homomorphism, if it has been defined.

If x is not None, return the value of the lift morphism on x.

**EXAMPLES:**

```python
sage: R.<x,y> = QQ[]
sage: f = R.hom([x,x])
sage: f(x+y)
2*x
sage: f.lift()
Traceback (most recent call last):
...  ValueError: no lift map defined
sage: g = R.hom(R)
sage: f._set_lift(g)
sage: f.lift() == g
True
sage: f.lift(x)
x
```

**pushforward**(I)

Returns the pushforward of the ideal I under this ring homomorphism.

**EXAMPLES:**

```python
sage: R.<x,y> = QQ[]; S.<xx,yy> = R.quo([x^2, y^2]); f = S.cover()  # needs sage.libs.singular
sage: f.pushforward(R.ideal([x, 3*x + x*y + y^2]))  # needs sage.libs.singular
```

(continues on next page)
class sage.rings.morphism.RingHomomorphism_cover

Bases: RingHomomorphism

A homomorphism induced by quotienting a ring out by an ideal.

EXAMPLES:

```python
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = R.quo(x^2 + y^2)

# needs sage.libs.singular
sage: phi = S.cover(); phi
Ring morphism:
  From: Multivariate Polynomial Ring in x, y over Rational Field
  To:   Quotient of Multivariate Polynomial Ring in x, y over Rational Field
        by the ideal (x^2 + y^2)
  Defn: Natural quotient map
sage: phi(x + y)
a + b
```

kernel()

Return the kernel of this covering morphism, which is the ideal that was quotiented out by.

EXAMPLES:

```python
sage: f = Zmod(6).cover()
sage: f.kernel()
Principal ideal (6) of Integer Ring
```

class sage.rings.morphism.RingHomomorphism_from_base

Bases: RingHomomorphism

A ring homomorphism determined by a ring homomorphism of the base ring.

AUTHOR:

• Simon King (initial version, 2010-04-30)

EXAMPLES:

We define two polynomial rings and a ring homomorphism:

```python
sage: R.<x,y> = QQ[]
sage: S.<z> = QQ[]
sage: f = R.hom([2*z,3*z],S)
```

Now we construct polynomial rings based on R and S, and let f act on the coefficients:

```python
sage: PR.<t> = R[]
sage: PS = S['t']
sage: Pf = PR.hom(f,PS)
sage: Pf
```
Ring morphism:
- From: Univariate Polynomial Ring in \( t \)
- over Multivariate Polynomial Ring in \( x, y \) over Rational Field
- To: Univariate Polynomial Ring in \( t \)
- over Univariate Polynomial Ring in \( z \) over Rational Field
- Defn: Induced from base ring by
  Ring morphism:
   - From: Multivariate Polynomial Ring in \( x, y \) over Rational Field
   - To: Univariate Polynomial Ring in \( z \) over Rational Field
   - Defn: \( x \rightarrow 2*z \)
     \( y \rightarrow 3*z \)

\[
p = (x - 4*y + 1/13)*t^2 + (1/2*x^2 - 1/3*y^2)*t + 2*y^2 + x
\]

\[
Pf(p)
\]
\[
(-10*z + 1/13)*t^2 - z^2 + 18*z^2 + 2*z
\]

Similarly, we can construct the induced homomorphism on a matrix ring over our polynomial rings:

\[
\text{sage:} \quad \text{# needs sage.modules}
\]
\[
\text{sage:} \quad R = \text{MatrixSpace}(\mathbb{R}, 2, 2)
\]
\[
\text{sage:} \quad S = \text{MatrixSpace}(\mathbb{S}, 2, 2)
\]
\[
\text{sage:} \quad M = R([x^2 + 1/7*x*y - y^2, -1/2*y^2 + 2*y + 1/6,
\text{.....}
\quad 4*x^2 - 14*x, 1/2*y^2 + 13/4*x - 2/11*y])
\]
\[
\text{sage:} \quad f = R.hom(f, S)
\]
\[
\text{sage:} \quad Mf(M)
\]
\[
\begin{bmatrix}
-29/7*z^2 & -9/2*z^2 + 6*z + 1/6 \\
16*z^2 - 28*z & 9/2*z^2 + 131/22*z
\end{bmatrix}
\]

The construction of induced homomorphisms is recursive, and so we have:

\[
\text{sage:} \quad \text{# needs sage.modules}
\]
\[
\text{sage:} \quad PR = \text{MatrixSpace}(\mathbb{PR}, 2)
\]
\[
\text{sage:} \quad PS = \text{MatrixSpace}(\mathbb{PS}, 2)
\]
\[
\text{sage:} \quad M = PR([(-x + y)^2*t^2 + 58*t - 3*x^2 + x*y,
\text{.....}
\quad (- 1/7*x^2*y - 1/40*x)*t^2 + (5*x^2 + y^2)*t + 2*y,
\text{.....}
\quad (- 1/3*y + 1)^2 + 1/3*x^2*y + y^2 + 5/2*y + 1/4,
\text{.....}
\quad (x + 6*y + 1)^2])
\]
\[
\text{sage:} \quad f = PR.hom(f, PS); f
\]
\[
\text{Ring morphism:}
- From: Full MatrixSpace of 2 by 2 dense matrices over Univariate Polynomial
  Ring in \( t \) over Multivariate Polynomial Ring in \( x, y \) over Rational Field
- To: Full MatrixSpace of 2 by 2 dense matrices over Multivariate Polynomial Ring in \( x, y \) over Rational Field
- Defn: Induced from base ring by
  - Ring morphism:
    - From: Multivariate Polynomial Ring in \( x, y \) over Rational Field
    - To: Univariate Polynomial Ring in \( z \) over Rational Field
    - Defn: \( x \rightarrow 2*z \)
      \( y \rightarrow 3*z \)

\[
\text{sage:} \quad Mf(M)
\]
\[
\begin{bmatrix}
-10*z + 1/13 & -9/2*z^2 + 6*z + 1/6 \\
16*z^2 - 28*z & 9/2*z^2 + 131/22*z
\end{bmatrix}
\]
To: Full MatrixSpace of 2 by 2 dense matrices over Univariate Polynomial
Ring in t over Univariate Polynomial Ring in z over Rational Field
Defn: Induced from base ring by
Ring morphism:
From: Univariate Polynomial Ring in t
over Multivariate Polynomial Ring in x, y over Rational Field
To: Univariate Polynomial Ring in t
over Univariate Polynomial Ring in z over Rational Field
Defn: Induced from base ring by
Ring morphism:
From: Multivariate Polynomial Ring in x, y over Rational Field
To: Univariate Polynomial Ring in z over Rational Field
Defn: x |--> 2*z
y |--> 3*z

sage: MPf(M)
\[
\begin{bmatrix}
z*t^2 + 58*t - 6*z^2 & (-6/7*z^2 - 1/20*z)*t^2 + 29*z^2*t + 6*z \\
(-z + 1)*t^2 + 11*z^2 + 15/2*z + 1/4 & (20*z + 1)*t^2
\end{bmatrix}
\]

inverse()
Return the inverse of this ring homomorphism if the underlying homomorphism of the base ring is invertible.

EXAMPLES:

sage: R.<x,y> = QQ[]
sage: S.<a,b> = QQ[]
sage: f = R.hom([a + b, a - b], S)
sage: PR.<t> = R[]
sage: PS = S['t']
sage: Pf = PR.hom(f, PS)
sage: Pf.inverse()

... needs sage.libs.singular

Ring morphism:
From: Univariate Polynomial Ring in t over Multivariate Polynomial Ring in a, b over Rational Field
To: Univariate Polynomial Ring in t over Multivariate Polynomial Ring in x, y over Rational Field
Defn: Induced from base ring by
Ring morphism:
From: Multivariate Polynomial Ring in a, b over Rational Field
To: Multivariate Polynomial Ring in x, y over Rational Field
Defn: a |--> 1/2*x + 1/2*y
b |--> 1/2*x - 1/2*y
sage: Pf.inverse()(Pf(x*t^2 + y*t))

... needs sage.libs.singular

underlying_map()
Return the underlying homomorphism of the base ring.

EXAMPLES:
class sage.rings.morphism.RingHomomorphism_from_fraction_field
Bases: RingHomomorphism

Morphisms between fraction fields.

inverse()
Return the inverse of this ring homomorphism if it exists.

EXAMPLES:

```python
sage: S.<x> = QQ[]
sage: f = S.hom([2*x - 1])
sage: g = f.extend_to_fraction_field()  # need sage.libs.singular
sage: g.inverse()  # need sage.libs.singular
```

Ring endomorphism of Fraction Field of Univariate Polynomial Ring in x over Rational Field
Defn: x |--> 1/2*x + 1/2

class sage.rings.morphism.RingHomomorphism_from_quotient
Bases: RingHomomorphism

A ring homomorphism with domain a generic quotient ring.

INPUT:

- `parent` – a ring homset \( \text{Hom}(R, S) \)
- `phi` – a ring homomorphism \( C \to S \), where \( C \) is the domain of \( R.\text{cover}() \)

OUTPUT: a ring homomorphism

The domain \( R \) is a quotient object \( C \to R \), and \( R.\text{cover}() \) is the ring homomorphism \( \varphi : C \to R \). The condition on the elements \( \text{im}_\text{gens} \) of \( S \) is that they define a homomorphism \( C \to S \) such that each generator of the kernel of \( \varphi \) maps to 0.

EXAMPLES:

```python
sage: # need sage.libs.singular
sage: R.<x, y, z> = PolynomialRing(QQ, 3)
sage: S.<a, b, c> = R.quo(x^3 + y^3 + z^3)
sage: phi = S.hom([b, c, a]); phi
```

Ring endomorphism of Quotient of Multivariate Polynomial Ring in x, y, z
over Rational Field by the ideal \((x^3 + y^3 + z^3)\)
Defn: a |--> b
   b |--> c

(continues on next page)
c |--> a
sage: phi(a + b + c)
a + b + c
sage: loads(dumps(phi)) == phi
True

Validity of the homomorphism is determined, when possible, and a TypeError is raised if there is no homomorphism sending the generators to the given images:

sage: S.hom([b^2, c^2, a^2])  # needs sage.libs.singular
Traceback (most recent call last):
...:
ValueError: relations do not all (canonically) map to 0 under map determined by images of generators

**morphism_from_cover()**

Underlying morphism used to define this quotient map, i.e., the morphism from the cover of the domain.

**EXAMPLES:**

sage: R.<x,y> = QQ[]; S.<xx,yy> = R.quo([x^2, y^2])  # needs sage.libs.singular
sage: S.hom([yy,xx]).morphism_from_cover()  # needs sage.libs.singular
Ring morphism:
From: Multivariate Polynomial Ring in x, y over Rational Field
To:   Quotient of Multivariate Polynomial Ring in x, y
       over Rational Field by the ideal (x^2, y^2)
Defn: x |--> yy
      y |--> xx

**class** sage.rings.morphism.RingHomomorphism_im_gens

**Bases:** RingHomomorphism

A ring homomorphism determined by the images of generators.

**base_map()**

Return the map on the base ring that is part of the defining data for this morphism. May return None if a coercion is used.

**EXAMPLES:**

sage: # needs sage.rings.number_field
sage: R.<x> = ZZ[]
sage: K.<i> = NumberField(x^2 + 1)
sage: cc = K.hom([-i])
sage: S.<y> = K[]
sage: phi = S.hom([y^2], base_map=cc)
sage: phi
Ring endomorphism of Univariate Polynomial Ring in y
over Number Field in i with defining polynomial x^2 + 1
Defn: y |--> y^2
      with map of base ring

(continues on next page)
sage: phi(y)
y^2
sage: phi(i*y)
-i*y^2
sage: phi.base_map()
Composite map:
  From: Number Field in i with defining polynomial x^2 + 1
  To: Univariate Polynomial Ring in y over Number Field in i
       with defining polynomial x^2 + 1
  Defn: Ring endomorphism of Number Field in i with defining polynomial x^2 +
-1
  then
  Polynomial base injection morphism:
  From: Number Field in i with defining polynomial x^2 + 1
  To: Univariate Polynomial Ring in y over Number Field in i
       with defining polynomial x^2 + 1

im_gens()

Return the images of the generators of the domain.

OUTPUT:

• list – a copy of the list of gens (it is safe to change this)

EXAMPLES:

```
sage: R.<x,y> = QQ[

```

We verify that the returned list of images of gens is a copy, so changing it doesn’t change f:

```
sage: f.im_gens()[0] = 5
sage: f.im_gens()
[x, x + y]
```

class sage.rings.morphism.RingMap

Bases: Morphism

Set-theoretic map between rings.

class sage.rings.morphism.RingMap_lift

Bases: RingMap

Given rings $R$ and $S$ such that for any $x \in R$ the function $x.lift()$ is an element that naturally coerces to $S$, this returns the set-theoretic ring map $R \to S$ sending $x$ to $x.lift()$.

EXAMPLES:

```
sage: R.<x,y> = QQ[

```

(continues on next page)
Set-theoretic ring morphism:
From: Quotient of Multivariate Polynomial Ring in x, y
over Rational Field by the ideal (x^2 + y^2, y)
To: Multivariate Polynomial Ring in x, y over Rational Field
Defn: Choice of lifting map

sage: S.lift() == 0
False

Since github issue #11068, it is possible to create quotient rings of non-commutative rings by two-sided ideals.
It was needed to modify `RingMap_lift` so that rings can be accepted that are no instances of `sage.rings.ring.Ring`, as in the following example:

```
sage: # needs sage.modules sage.rings.finite_rings
sage: MS = MatrixSpace(GF(5), 2, 2)
sage: I = MS * [MS.0*MS.1, MS.2+MS.3] * MS
sage: Q = MS.quo(I)
sage: Q.0*Q.1
[0 1]
[0 0]
```

### 3.2 Space of homomorphisms between two rings

`sage.rings.homset.RingHomset(R, S, category=None)`

Construct a space of homomorphisms between the rings R and S.

For more on homsets, see `Hom()`.

**EXAMPLES:**

```
sage: Hom(ZZ, QQ) # indirect doctest
Set of Homomorphisms from Integer Ring to Rational Field
```

**class** `sage.rings.homset.RingHomset_generic(R, S, category=None)`

Bases: `HomsetWithBase`

A generic space of homomorphisms between two rings.

**EXAMPLES:**

```
sage: Hom(ZZ, QQ)
Set of Homomorphisms from Integer Ring to Rational Field
sage: QQ.Hom(ZZ)
Set of Homomorphisms from Rational Field to Integer Ring
```

**Element**

alias of `RingHomomorphism`

**has_coerce_map_from**(x)

The default for coercion maps between ring homomorphism spaces is very restrictive (until more implementation work is done).
Currently this checks if the domains and the codomains are equal.

EXAMPLES:

```python
sage: H = Hom(ZZ, QQ)
sage: H2 = Hom(QQ, ZZ)
sage: H.has_coerce_map_from(H2)
False
```

natural_map()

Returns the natural map from the domain to the codomain.

The natural map is the coercion map from the domain ring to the codomain ring.

EXAMPLES:

```python
sage: H = Hom(ZZ, QQ)
sage: H.natural_map()
Natural morphism:
  From: Integer Ring
  To:   Rational Field
```

zero()

Return the zero element of this homset.

EXAMPLES:

Since a ring homomorphism maps 1 to 1, there can only be a zero morphism when mapping to the trivial ring:

```python
sage: Hom(ZZ, Zmod(1)).zero()
Ring morphism:
  From: Integer Ring
  To:   Ring of integers modulo 1
        Defn: 1 |--> 0
sage: Hom(ZZ, Zmod(2)).zero()
Traceback (most recent call last):
  ... ValueError: homset has no zero element
```

class sage.rings.homset.RingHomset_quo_ring(R, S, category=None)

Bases: RingHomset_generic

Space of ring homomorphisms where the domain is a (formal) quotient ring.

EXAMPLES:

```python
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = R.quotient(x^2 + y^2)
  # needs sage.libs.singular
sage: phi = S.hom([b,a]); phi
  # needs sage.libs.singular
Ring endomorphism of Quotient of Multivariate Polynomial Ring in x, y
over Rational Field by the ideal (x^2 + y^2)
  Defn: a |--> b
  b |--> a
sage: phi(a)
  # needs sage.libs.singular
```

(continues on next page)
Element

alias of RingHomomorphism_from_quotient

sage.rings.homset.is_RingHomset(H)

Return True if H is a space of homomorphisms between two rings.

EXAMPLES:

```python
sage: from sage.rings.homset import is_RingHomset as is_RH
sage: is_RH(Hom(ZZ, QQ))
True
sage: is_RH(ZZ)
False
sage: is_RH(Hom(RR, CC))  # needs sage.rings.real_mpfr
True
sage: is_RH(Hom(FreeModule(ZZ,1), FreeModule(QQ,1)))  # needs sage.modules
False
```
4.1 Quotient Rings

AUTHORS:

- William Stein
- Simon King (2011-04): Put it into the category framework, use the new coercion model.
- Simon King (2011-04): Quotients of non-commutative rings by twosided ideals.

Todo: The following skipped tests should be removed once github issue #13999 is fixed:

```
sage: TestSuite(S).run(skip=['_test_nonzero_equal', '_test_elements', '_test_zero'])
```

In github issue #11068, non-commutative quotient rings \( R/I \) were implemented. The only requirement is that the two-sided ideal \( I \) provides a reduce method so that \( I.\text{reduce}(x) \) is the normal form of an element \( x \) with respect to \( I \) (i.e., we have \( I.\text{reduce}(x) = I.\text{reduce}(y) \) if \( x - y \in I \), and \( x - I.\text{reduce}(x) \in I \)). Here is a toy example:

```
sage: from sage.rings.noncommutative_ideals import Ideal_nc
sage: from itertools import product
sage: class PowerIdeal(Ideal_nc):
....:     def __init__(self, R, n):
....:         self._power = n
....:         Ideal_nc.__init__(self, R, [R.prod(m) for m in product(R.gens(), repeat=n)])
....:     def reduce(self, x):
....:         R = self.ring()
....:         return add([c*R(m) for m, c in x if len(m)<self._power], R(0))
sage: F.<x,y,z> = FreeAlgebra(QQ, 3)  # needs sage.combinat sage.modules
sage: I3 = PowerIdeal(F,3); I3
Twosided Ideal (x^3, x^2*y, x^2*z, x*y*x, x*y^2, x*y*z, x*z*x, x*z^2, y*x^2, y*x*y, y*x*z, y^2*x, y^2*z, y*z*x, y*z^2, z*x^2, z*x*y, z*x*z, z*y*x, z*y^2, z*y*z, z^2*x, z^2*y, z^3) of Free Algebra on 3 generators (x, y, z) over Rational Field
```

Free algebras have a custom quotient method that serves at creating finite dimensional quotients defined by multiplication matrices. We are bypassing it, so that we obtain the default quotient:
sage: # needs sage.combinat sage.modules
sage: Q3.<a,b,c> = F.quotient(I3)
sage: Q3
Quotient of Free Algebra on 3 generators (x, y, z) over Rational Field by
the ideal (x^3, x^2*y, x^2*z, x*y^2, x*y*z, x*z^2, y*x^2, y*x*y, y*x*z, y^2*x, y^3, y^2*z, y*z^2, y^2*y, y*z^2, z*x^2, z*x*y, z*y^2, z*y*z, z^2*x, z^2*y, z^3)
sage: (a+b+2)^4
16 + 32*a + 32*b + 24*a^2 + 24*a*b + 24*b*a + 24*b^2
sage: Q3.is_commutative()
False

Even though \(Q_3\) is not commutative, there is commutativity for products of degree three:

sage: a*(b*c)-(b*c)*a==F.zero()  # needs sage.combinat sage.modules
True

If we quotient out all terms of degree two then of course the resulting quotient ring is commutative:

sage: # needs sage.combinat sage.modules
sage: I2 = PowerIdeal(F,2); I2
Twosided Ideal (x^2, x*y, x*z, y*x, y^2, y*z, z*x, z*y, z^2) of Free Algebra
on 3 generators (x, y, z) over Rational Field
sage: Q2.<a,b,c> = F.quotient(I2)
sage: Q2.is_commutative()
True
sage: (a+b+2)^4
16 + 32*a + 32*b

Since github issue #7797, there is an implementation of free algebras based on Singular’s implementation of the Letterplace Algebra. Our letterplace wrapper allows to provide the above toy example more easily:

sage: # needs sage.combinat sage.libs.singular sage.modules
sage: from itertools import product
sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
sage: Q3 = F.quo(F*[F.prod(m) for m in product(F.gens(), repeat=3)]*F)
sage: Q3
Quotient of Free Associative Unital Algebra on 3 generators (x, y, z)
over Rational Field by the ideal (x*x*x, x*x*y, x*x*z, x*y*x, x*y*y, x*y*z, 
x*z*x, x*z*y, x*z*z, y*x*x, y*x*y, y*x*z, y*y*x, y*y*y, y*y*z, y*z*x, y*z*y, 
y*z*z, z*x*x, z*x*y, z*x*z, z*y*x, z*y*y, z*y*z, z*z*x, z*z*y, z*z*z)
sage: Q3.0*Q3.1 - Q3.1*Q3.0
xbar*ybar - ybar*xbar
sage: Q3.0*(Q3.1*Q3.2) - (Q3.1*Q3.2)*Q3.0
0
sage: Q2 = F.quo(F*[F.prod(m) for m in product(F.gens(), repeat=2)]*F)
sage: Q2.is_commutative()
True

\texttt{sage.rings.quotient_ring.QuotientRing}(R, I, names=None, **kwds)

Creates a quotient ring of the ring \(R\) by the twosided ideal \(I\).

Variables are labeled by \texttt{names} (if the quotient ring is a quotient of a polynomial ring). If \texttt{names} isn’t given,
‘bar’ will be appended to the variable names in $R$.

**INPUT:**

- $R$ – a ring.
- $I$ – a twosided ideal of $R$.
- `names` – (optional) a list of strings to be used as names for the variables in the quotient ring $R/I$.
- further named arguments that will be passed to the constructor of the quotient ring instance.

**OUTPUT:** $R/I$ - the quotient ring $R$ mod the ideal $I$

**ASSUMPTION:**

$I$ has a method $I.reduce(x)$ returning the normal form of elements $x \in R$. In other words, it is required that $I.reduce(x) == I.reduce(y) \iff x - y \in I$, and $x - I.reduce(x)$ in $I$, for all $x, y \in R$.

**EXAMPLES:**

Some simple quotient rings with the integers:

```
sage: R = QuotientRing(ZZ, 7*ZZ); R
Quotient of Integer Ring by the ideal (7)
sage: R.gens()
(1,)
sage: 1*R(3); 6*R(3); 7*R(3)
3
4
0
```

```
sage: S = QuotientRing(ZZ, ZZ.ideal(8)); S
Quotient of Integer Ring by the ideal (8)
sage: 2*S(4)
0
```

With polynomial rings (note that the variable name of the quotient ring can be specified as shown below):

```
sage: # needs sage.libs.pari
sage: P.<x> = QQ[]
sage: R.<xx> = QuotientRing(P, P.ideal(x^2 + 1))
sage: R
Univariate Quotient Polynomial Ring in xx over Rational Field with modulus x^2 + 1
sage: R.gens(); R.gen()
(xx,)
xx
sage: for n in range(4): xx^n
1
xx
-1
-xx
```

```
sage: # needs sage.libs.pari
sage: P.<x> = QQ[]
sage: S = QuotientRing(P, P.ideal(x^2 - 2))
sage: S
```

(continues on next page)
Univariate Quotient Polynomial Ring in xbar over Rational Field
with modulus x^2 - 2
sage: xbar = S.gen(); S.gen()
xbar
sage: for n in range(3): xbar^n
1
xbar
2

Sage coerces objects into ideals when possible:

```
sage: P.<x> = QQ[]
sage: R = QuotientRing(P, x^2 + 1); R
# needs sage.libs.pari
Univariate Quotient Polynomial Ring in xbar over Rational Field
with modulus x^2 + 1
```

By Noether’s homomorphism theorems, the quotient of a quotient ring of \( R \) is just the quotient of \( R \) by the sum of the ideals. In this example, we end up modding out the ideal \((x)\) from the ring \( \mathbb{Q}[x,y]\):

```
sage: # needs sage.libs.pari sage.libs.singular
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = QuotientRing(R, R.ideal(1 + y^2))
sage: T.<c,d> = QuotientRing(S, S.ideal(a))
sage: T
Quotient of Multivariate Polynomial Ring in x, y over Rational Field
by the ideal (x, y^2 + 1)
sage: R.gens(); S.gens(); T.gens()
(x, y)
(a, b)
(θ, d)
sage: for n in range(4): d^n
1
d
-1
-d
```

**class** `sage.rings.quotient_ring.QuotientRingIdeal_generic` *(ring, gens, coerce=True)*

Bases: *Ideal_generic*

Specialized class for quotient-ring ideals.

**EXAMPLES:**

```
sage: Zmod(9).ideal([-6,9])
Ideal (3, θ) of Ring of integers modulo 9
```

**class** `sage.rings.quotient_ring.QuotientRingIdeal_principal` *(ring, gens, coerce=True)*

Bases: *Ideal_principal, QuotientRingIdeal_generic*

Specialized class for principal quotient-ring ideals.

**EXAMPLES:**
class `sage.rings.quotient_ring.QuotientRing_generic(R, I, names, category=None)`

Bases: `QuotientRing_nc`, `CommutativeRing`

Creates a quotient ring of a *commutative* ring \( R \) by the ideal \( I \).

EXAMPLES:

```python
sage: R.<x> = PolynomialRing(ZZ)
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: S = R.quotient_ring(I); S
Quotient of Univariate Polynomial Ring in x over Integer Ring
by the ideal (x^2 + 3*x + 4, x^2 + 1)
```

class `sage.rings.quotient_ring.QuotientRing_nc(R, I, names, category=None)`

Bases: `Ring`, `ParentWithGens`

The quotient ring of \( R \) by a twosided ideal \( I \).

This class is for rings that do not inherit from `CommutativeRing`.

EXAMPLES:

Here is a quotient of a free algebra by a twosided homogeneous ideal:

```python
sage: # needs sage.combinat sage.libs.singular sage.modules
sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
sage: I = F * [x*y + y*z, x^2 + x*y - y*x - y^2]*F
sage: Q.<a,b,c> = F.quo(I); Q
Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) over Rational Field by the ideal (x*y + y*z, x*x + x*y - y*x - y*y)
sage: a*b
-b*c
sage: a^3
-b*c*a - b*b*c - b*c*c
```

A quotient of a quotient is just the quotient of the original top ring by the sum of two ideals:

```python
sage: # needs sage.combinat sage.libs.singular sage.modules
sage: J = Q * [a^3 - b^3] * Q
sage: R.<i,j,k> = Q.quo(J); R
Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) over Rational Field by the ideal (-y*z^2 + y*x^2 - 2*y*z*z, x*y + y*z, x*x + x*y - y*x - y*y)
sage: i^3
-j*k*i - j*k*j - j*k*k
sage: j^3
-j*k*i - j*k*j - j*k*k
```

For rings that *do* inherit from `CommutativeRing`, we provide a subclass `QuotientRing_generic`, for backwards compatibility.

EXAMPLES:
General Rings, Ideals, and Morphisms, Release 10.2

sage: R.<x> = PolynomialRing(ZZ, 'x')
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: S = R.quotient_ring(I); S
Quotient of Univariate Polynomial Ring in x over Integer Ring
by the ideal (x^2 + 3*x + 4, x^2 + 1)

sage: R.<x,y> = PolynomialRing(QQ)
sage: S.<a,b> = R.quo(x^2 + y^2)
# needs sage.libs.singular
sage: a^2 + b^2 == 0
needs sage.libs.singular
True
sage: S(0) == a^2 + b^2
needs sage.libs.singular
True

Again, a quotient of a quotient is just the quotient of the original top ring by the sum of two ideals.

sage: # needs sage.libs.singular
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = R.quo(1 + y^2)
sage: T.<c,d> = S.quo(a)
sage: T
Quotient of Multivariate Polynomial Ring in x, y over Rational Field
by the ideal (x, y^2 + 1)
sage: T.gens()
(0, d)

Element

alias of QuotientRingElement

ambient()

Returns the cover ring of the quotient ring: that is, the original ring \( R \) from which we modded out an ideal, \( I \).

EXAMPLES:

sage: Q = QuotientRing(ZZ, 7 * ZZ)
sage: Q.cover_ring()
Integer Ring

sage: P.<x> = QQ[]
sage: Q = QuotientRing(P, x^2 + 1)
# needs sage.libs.pari
sage: Q.cover_ring()
# needs sage.libs.pari
Univariate Polynomial Ring in x over Rational Field

characteristic()

Return the characteristic of the quotient ring.

Todo: Not yet implemented!
EXAMPLES:

```python
sage: Q = QuotientRing(ZZ, 7*ZZ)
sage: Q.characteristic()
Traceback (most recent call last):
  ... NotImplementedError
```

**construction()**

Returns the functorial construction of self.

**EXAMPLES:**

```python
sage: R.<x> = PolynomialRing(ZZ, 'x')
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: R.quotient_ring(I).construction()
(QuotientFunctor, Univariate Polynomial Ring in x over Integer Ring)
sage: # needs sage.combinat sage.libs.singular sage.modules
sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
sage: I = F * [x*y + y*z, x^2 + x*y - y*x - y^2] * F
sage: Q = F.quot(I)
sage: Q.construction()
(QuotientFunctor,
 Free Associative Unital Algebra on 3 generators (x, y, z) over Rational Field)
```

**cover()**

The covering ring homomorphism \( R \to R/I \), equipped with a section.

**EXAMPLES:**

```python
sage: R = ZZ.quo(3 * ZZ)
sage: pi = R.cover()
sage: pi
Ring morphism:
  From: Integer Ring
  To:   Ring of integers modulo 3
  Defn: Natural quotient map
sage: pi(5)
2
sage: l = pi.lift()
sage: l
```

```python
sage: # needs sage.libs.singular
sage: R.<x,y> = PolynomialRing(QQ)
sage: Q = R.quot((x^2, y^2))
sage: pi = Q.cover()
sage: pi(x^3 + y)
ybar
sage: l = pi.lift(x + y^3)
```

(continues on next page)

4.1. Quotient Rings
by the ideal \((x^2, y^2)\)

To: Multivariate Polynomial Ring in \(x, y\) over Rational Field

Defn: Choice of lifting map

\[
\text{sage: } l(x + y^3) \\quad x
\]

\textbf{cover\textunderscore ring()}

Returns the cover ring of the quotient ring: that is, the original ring \(R\) from which we modded out an ideal, \(I\).

\textbf{EXAMPLES:}

\[
\text{sage: } Q = \text{QuotientRing}(\text{ZZ}, 7 \cdot \text{ZZ})
\]

\[
\text{sage: } Q.\text{cover\textunderscore ring}()
\]

\text{Integer Ring}

\[
\text{sage: } P.\langle x \rangle = \text{QQ}[x]
\]

\[
\text{sage: } Q = \text{QuotientRing}(P, x^2 + 1)
\]

\[
\# \text{needs sage\textunderscore libs\textunderscore pari}
\]

\[
\text{sage: } Q.\text{cover\textunderscore ring}()
\]

\[
\# \text{needs sage\textunderscore libs\textunderscore pari}
\]

\text{Univariate Polynomial Ring in } x \text{ over Rational Field}

\textbf{defining\textunderscore ideal()}

Returns the ideal generating this quotient ring.

\textbf{EXAMPLES:}

In the integers:

\[
\text{sage: } Q = \text{QuotientRing}(\text{ZZ}, 7 \cdot \text{ZZ})
\]

\[
\text{sage: } Q.\text{defining\textunderscore ideal}()
\]

\text{Principal ideal (7) of Integer Ring}

An example involving a quotient of a quotient. By Noether’s homomorphism theorems, this is actually a quotient by a sum of two ideals:

\[
\text{sage: } # \text{needs sage\textunderscore libs\textunderscore singular}
\]

\[
\text{sage: } R.\langle x, y \rangle = \text{PolynomialRing}(\text{QQ}, 2)
\]

\[
\text{sage: } S.\langle a, b \rangle = \text{QuotientRing}(R, R.\text{ideal}(1 + y^2))
\]

\[
\text{sage: } T.\langle c, d \rangle = \text{QuotientRing}(S, S.\text{ideal}(a))
\]

\[
\text{sage: } S.\text{defining\textunderscore ideal}()
\]

\text{Ideal (y^2 + 1) of Multivariate Polynomial Ring in x, y over Rational Field}

\[
\text{sage: } T.\text{defining\textunderscore ideal}()
\]

\text{Ideal (x, y^2 + 1) of Multivariate Polynomial Ring in x, y over Rational Field}

\textbf{gen(i=0)}

Returns the \(i\)-th generator for this quotient ring.

\textbf{EXAMPLES:}

\[
\text{sage: } R = \text{QuotientRing}(\text{ZZ}, 7 \cdot \text{ZZ})
\]

\[
\text{sage: } R.\text{gen}(0)
\]

\(1\)
sage: # needs sage.libs.singular
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = QuotientRing(R, R.ideal(1 + y^2))
sage: T.<c,d> = QuotientRing(S, S.ideal(a))
sage: T
Quotient of Multivariate Polynomial Ring in x, y over Rational Field
by the ideal (x, y^2 + 1)
sage: R.gen(0); R.gen(1)
x
y
sage: S.gen(0); S.gen(1)
a
b
sage: T.gen(0); T.gen(1)
0
d

ideal(*gens, **kwds)

Return the ideal of self with the given generators.

EXAMPLES:

sage: R.<x,y> = PolynomialRing(QQ)
sage: S = R.quotient_ring(x^2 + y^2)
sage: S.ideal()
Ideal (0) of Quotient of Multivariate Polynomial Ring in x, y
over Rational Field by the ideal (x^2 + y^2)
sage: S.ideal(x + y + 1)
Ideal (xbar + ybar + 1) of Quotient of Multivariate Polynomial Ring in x, y
over Rational Field by the ideal (x^2 + y^2)

is_commutative()

Tell whether this quotient ring is commutative.

Note: This is certainly the case if the cover ring is commutative. Otherwise, if this ring has a finite number of generators, it is tested whether they commute. If the number of generators is infinite, a
NotImplementedError is raised.

AUTHOR:

• Simon King (2011-03-23): See github issue #7797.

EXAMPLES:

Any quotient of a commutative ring is commutative:

sage: P.<a,b,c> = QQ[]
sage: P.quo(P.random_element()).is_commutative()
True

The non-commutative case is more interesting:
In the next example, the generators apparently commute:

```python
sage: # needs sage.combinat sage.libs.singular sage.modules
sage: J = F * [x*y - y*x, x*z - z*x, y*z - z*y, x^3 - y^3] * F
sage: R = F.quo(J)
sage: R.is_commutative()
True
```

**is_field** *(proof=True)*

Returns `True` if the quotient ring is a field. Checks to see if the defining ideal is maximal.

**is_integral_domain** *(proof=True)*

With `proof` equal to `True` (the default), this function may raise a `NotImplementedError`. When `proof` is `False`, if `True` is returned, then `self` is definitely an integral domain. If the function returns `False`, then either `self` is not an integral domain or it was unable to determine whether or not `self` is an integral domain.

**EXAMPLES:**

```python
sage: R.<x,y> = QQ[]
sage: R.quo(x^2 - y).is_integral_domain() # needs sage.libs.singular
True
sage: R.quo(x^2 - y^2).is_integral_domain() # needs sage.libs.singular
False
sage: R.<a,b,c> = ZZ[]
sage: Q = R.quotient_ring([a, b])
sage: Q.is_integral_domain() # needs sage.libs.singular
Traceback (most recent call last):
  ... raise NotImplementedError
NotImplementedError
sage: Q.is_integral_domain(proof=False)
False
```

**is_noetherian()**

Return `True` if this ring is Noetherian.

**EXAMPLES:**

```python
sage: R = QuotientRing(ZZ, 102 * ZZ)
sage: R.is_noetherian()
```

(continues on next page)
True

```
sage: P.<x> = QQ[]
sage: R = QuotientRing(P, x^2 + 1)
```

needs sage.libs.pari

```
sage: R.is_noetherian()
True
```

If the cover ring of `self` is not Noetherian, we currently have no way of testing whether `self` is Noetherian, so we raise an error:

```
sage: R.<x> = InfinitePolynomialRing(QQ)
sage: R.is_noetherian()
False
```

```
sage: I = R.ideal([x[1]^2, x[2]])
sage: S = R.quotient(I)
sage: S.is_noetherian()
Traceback (most recent call last):
... NotImplementedError
```

**lift** *(x=None)*

Return the lifting map to the cover, or the image of an element under the lifting map.

**Note:** The category framework imposes that `Q.lift(x)` returns the image of an element `x` under the lifting map. For backwards compatibility, we let `Q.lift()` return the lifting map.

**EXAMPLES:**

```
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S = R.quotient(x^2 + y^2)
sage: S.lift()
```

needs sage.libs.singular

```
Set-theoretic ring morphism:
  From: Quotient of Multivariate Polynomial Ring in x, y over Rational Field
          by the ideal (x^2 + y^2)
  To:   Multivariate Polynomial Ring in x, y over Rational Field
        Defn: Choice of lifting map
```

```
sage: S.lift(S.0) == x
```

needs sage.libs.singular

```
True
```

**lifting_map**

Return the lifting map to the cover.

**EXAMPLES:**

```
sage: # needs sage.libs.singular
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S = R.quotient(x^2 + y^2)
sage: pi = S.cover(); pi
```

Ring morphism:
From: Multivariate Polynomial Ring in x, y over Rational Field
To: Quotient of Multivariate Polynomial Ring in x, y over Rational Field
by the ideal (x^2 + y^2)
Defn: Natural quotient map

```
sage: L = S.lifting_map(); L
Set-theoretic ring morphism:
From: Quotient of Multivariate Polynomial Ring in x, y over Rational Field
by the ideal (x^2 + y^2)
To: Multivariate Polynomial Ring in x, y over Rational Field
Defn: Choice of lifting map
```

```
sage: L(S.0)
x
sage: L(S.1)
y
```

Note that some reduction may be applied so that the lift of a reduction need not equal the original element:

```
sage: z = pi(x^3 + 2*y^2); z
˓→ needs sage.libs.singular
-xbar*ybar^2 + 2*ybar^2
sage: L(z)
˓→ needs sage.libs.singular
-x*y^2 + 2*y^2
sage: L(z) == x^3 + 2*y^2
˓→ needs sage.libs.singular
False
```

Test that there also is a lift for rings that are no instances of `Ring` (see github issue #11068):

```
sage: # needs sage.modules
sage: MS = MatrixSpace(GF(5), 2, 2)
sage: I = MS * [MS.0*MS.1, MS.2 + MS.3] * MS
sage: Q = MS.quo(I)
sage: Q.lift()
Set-theoretic ring morphism:
From: Quotient of Full MatrixSpace of 2 by 2 dense matrices
over Finite Field of size 5 by the ideal
( [0 1]
[0 0],

[0 0]
[1 1]
)
To: Full MatrixSpace of 2 by 2 dense matrices over Finite Field of size 5
Defn: Choice of lifting map
```

```
ngens()
Returns the number of generators for this quotient ring.
```

**Todo:** Note that `ngens` counts 0 as a generator. Does this make sense? That is, since 0 only generates
itself and the fact that this is true for all rings, is there a way to “knock it off” of the generators list if a generator of some original ring is modded out?

EXAMPLES:

```sage
R = QuotientRing(ZZ, 7*ZZ)
R.gens(); R.ngens()
(1,)
1

# needs sage.libs.singular
R.<x,y> = PolynomialRing(QQ,2)
S.<a,b> = QuotientRing(R, R.ideal(1 + y^2))
T.<c,d> = QuotientRing(S, S.ideal(a))
T
Quotient of Multivariate Polynomial Ring in x, y over Rational Field
by the ideal (x, y^2 + 1)
R.gens(); S.gens(); T.gens()
(x, y)
(a, b)
(0, d)
R.ngens(); S.ngens(); T.ngens()
2
2
2
```

`retract(x)`

The image of an element of the cover ring under the quotient map.

INPUT:

• `x` – An element of the cover ring

OUTPUT:

The image of the given element in `self`.

EXAMPLES:

```sage
R.<x,y> = PolynomialRing(QQ, 2)
S = R.quotient(x^2 + y^2)
S.retract((x+y)^2)
2*xbar*ybar
```

`term_order()`

Return the term order of this ring.

EXAMPLES:

```sage
P.<a,b,c> = PolynomialRing(QQ)
I = Ideal([a^2 - a, b^2 - b, c^2 - c])
Q = P.quotient(I)
Q.term_order()
Degree reverse lexicographic term order
```
4.2 Quotient Ring Elements

AUTHORS:
• William Stein

class sage.rings.quotient_ring_element.QuotientRingElement(parent, rep, reduce=True)

Bases: RingElement

An element of a quotient ring $R/I$.

INPUT:
• parent - the ring $R/I$
• rep - a representative of the element in $R$; this is used as the internal representation of the element
• reduce - bool (optional, default: True) - if True, then the internal representation of the element is rep reduced modulo the ideal $I$

EXAMPLES:

sage: R.<x> = PolynomialRing(ZZ)
sage: S.<xbar> = R.quotient_ring((4 + 3*x + x^2, 1 + x^2)); S
Quotient of Univariate Polynomial Ring in x over Integer Ring
by the ideal (x^2 + 3*x + 4, x^2 + 1)
sage: v = S.gens(); v
(xbar,)
sage: loads(v[0].dumps()) == v[0]
True
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S = R.quotient(x^2 + y^2); S
Quotient of Multivariate Polynomial Ring in x, y over Rational Field
by the ideal (x^2 + y^2)
sage: S.gens()  # needs sage.libs.singular
(xbar, ybar)

We name each of the generators.

sage: # needs sage.libs.singular
sage: S.<a,b> = R.quotient(x^2 + y^2)
sage: a
a
sage: b
b
sage: a^2 + b^2 == 0
True
sage: b.lift()
y
sage: (a^3 + b^2).lift()
-x*y^2 + y^2

is_unit()

Return True if self is a unit in the quotient ring.

EXAMPLES:

sage: R.<x,y> = QQ[]; S.<a,b> = R.quotient(1 - x*y); type(a)  # needs sage.libs.singular
<class 'sage.rings.quotient_ring.QuotientRing_generic_with_category.element_class'>
sage: a*b
1
sage: S(2).is_unit()  # needs sage.libs.singular
True

Check that github issue #29469 is fixed:

sage: a.is_unit()  # needs sage.libs.singular
True
sage: (a+b).is_unit()  # needs sage.libs.singular
False

lc()

Return the leading coefficient of this quotient ring element.

EXAMPLES:
sage: # needs sage.libs.singular
sage: R.<x,y,z> = PolynomialRing(GF(7), 3, order='lex')
sage: I = sage.rings.ideal.FieldIdeal(R)
sage: Q = R.quo(I)
sage: f = Q(z*y + 2*x)
sage: f.lc()
2

\textbf{lift()}  
If self is an element of $R/I$, then return self as an element of $R$.

\textbf{EXAMPLES:}

sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)  
# needs sage.libs.singular
<class 'sage.rings.quotient_ring.QuotientRing_generic_with_category.element_class'>
sage: a.lift()  
# needs sage.libs.singular
x
sage: (3/5*(a + a^2 + b^2)).lift()  
# needs sage.libs.singular
3/5*x

\textbf{lm()}  
Return the leading monomial of this quotient ring element.

\textbf{EXAMPLES:}

sage: # needs sage.libs.singular
sage: R.<x,y,z> = PolynomialRing(GF(7), 3, order='lex')
sage: I = sage.rings.ideal.FieldIdeal(R)
sage: Q = R.quo(I)
sage: f = Q(z*y + 2*x)
sage: f.lm()
xbar

\textbf{lt()}  
Return the leading term of this quotient ring element.

\textbf{EXAMPLES:}

sage: # needs sage.libs.singular
sage: R.<x,y,z> = PolynomialRing(GF(7), 3, order='lex')
sage: I = sage.rings.ideal.FieldIdeal(R)
sage: Q = R.quo(I)
sage: f = Q(z*y + 2*x)
sage: f.lt()
2*xbar

\textbf{monomials()}  
Return the monomials in self.

\textbf{OUTPUT:}

A list of monomials.
EXAMPLES:

```python
sage: # needs sage.libs.singular
sage: R.<x,y> = QQ[]; S.<a,b> = R.quotient(x^2 + y^2); type(a)
<class 'sage.rings.quotient_ring.QuotientRing_generic_with_category.element_class'>
sage: a.monomials()
[a]
sage: (a + a*b).monomials()
[a*b, a]
sage: R.zero().monomials()
[]
```

**reduce**

Reduce this quotient ring element by a set of quotient ring elements G.

**INPUT:**

- G - a list of quotient ring elements

**Warning:** This method is not guaranteed to return unique minimal results. For quotients of polynomial rings, use `reduce()` on the ideal generated by G, instead.

EXAMPLES:

```python
sage: # needs sage.libs.singular
sage: P.<a,b,c,d,e> = PolynomialRing(GF(2), 5, order='lex')
sage: I1 = ideal([a*b + c*d + 1, a*c*e + d*e, ...
....: a*b*e + c*e, b*c + c*d*e + 1])
sage: Q = P.quotient(sage.rings.ideal.FieldIdeal(P))
sage: I2 = ideal([Q(f) for f in I1.gens()])
sage: f = Q((a*b + c*d + 1)^2 + e)
sage: f.reduce(I2.gens())
ebar
```

Notice that the result above is not minimal:

```python
sage: I2.reduce(f) # needs sage.libs.singular
0
```

**variables**

Return all variables occurring in self.

**OUTPUT:**

A tuple of linear monomials, one for each variable occurring in self.

EXAMPLES:

```python
sage: # needs sage.libs.singular
sage: R.<x,y> = QQ[]; S.<a,b> = R.quotient(x^2 + y^2); type(a)
<class 'sage.rings.quotient_ring.QuotientRing_generic_with_category.element_class'>
sage: a.variables()
(continues on next page)
```
5.1 Fraction Field of Integral Domains

AUTHORS:

- William Stein (with input from David Joyner, David Kohel, and Joe Wetherell)
- Burcin Erocal
- Julian Rüth (2017-06-27): embedding into the field of fractions and its section

EXAMPLES:
Quotienting is a constructor for an element of the fraction field:

```
sage: R.<x> = QQ[]
sage: (x^2-1)/(x+1)
x - 1
sage: parent((x^2-1)/(x+1))
Fraction Field of Univariate Polynomial Ring in x over Rational Field
```

The GCD is not taken (since it doesn’t converge sometimes) in the inexact case:

```
sage: # needs sage.rings.real_mpfr
sage: Z.<z> = CC[]
sage: I = CC.gen()
sage: (1+I+z)/(z+0.1*I)
(z + 1.00000000000000 + I)/(z + 0.100000000000000*I)
sage: (1+I*z)/(z+1.1)
(I*z + 1.00000000000000)/(z + 1.10000000000000)
```

```
sage.rings.fraction_field.FractionField(R, names=None)
Create the fraction field of the integral domain R.

INPUT:

- R – an integral domain
- names – ignored

EXAMPLES:
We create some example fraction fields:
```
Dividing elements often implicitly creates elements of the fraction field:

```python
sage: x = PolynomialRing(RationalField(), 'x').gen()
sage: f = x/(x+1)
sage: g = x**3/(x+1)
sage: f/g
1/x^2
sage: g/f
x^2
```

The input must be an integral domain:

```python
sage: Frac(Integers(4))
Traceback (most recent call last):
  ...TypeError: R must be an integral domain.
```

```python
class sage.rings.fraction_field.FractionFieldEmbedding

Bases: DefaultConvertMap_unique

The embedding of an integral domain into its field of fractions.

EXAMPLES:

```python
sage: R.<x> = QQ[]
sage: f = R.fraction_field().coerce_map_from(R); f
Coercion map:
  From: Univariate Polynomial Ring in x over Rational Field
  To:   Fraction Field of Univariate Polynomial Ring in x over Rational Field

is_injective()

Return whether this map is injective.

EXAMPLES:

The map from an integral domain to its fraction field is always injective:

```python
sage: R.<x> = QQ[]
sage: R.fraction_field().coerce_map_from(R).is_injective()
True
```

is_surjective()

Return whether this map is surjective.

EXAMPLES:
```
sage: R.<x> = QQ[]
sage: R.fraction_field().coerce_map_from(R).is_surjective()
False
```

### section()

Return a section of this map.

**EXAMPLES:**

```
sage: R.<x> = QQ[]
sage: R.fraction_field().coerce_map_from(R).section()
Section map:
    From: Fraction Field of Univariate Polynomial Ring in x over Rational Field
    To:    Univariate Polynomial Ring in x over Rational Field
```

### class sage.rings.fraction_field.FractionFieldEmbeddingSection

Bases: `Section`

The section of the embedding of an integral domain into its field of fractions.

**EXAMPLES:**

```
sage: R.<x> = QQ[]
sage: f = R.fraction_field().coerce_map_from(R).section(); f
Section map:
    From: Fraction Field of Univariate Polynomial Ring in x over Rational Field
    To:    Univariate Polynomial Ring in x over Rational Field
```

### class sage.rings.fraction_field.FractionField_1poly_field

Bases: `FractionField_generic`

The fraction field of a univariate polynomial ring over a field.

Many of the functions here are included for coherence with number fields.

**class_number()**

Here for compatibility with number fields and function fields.

**EXAMPLES:**

```
sage: R.<t> = GF(5)[]; K = R.fraction_field()
sage: K.class_number()
1
```

**function_field()**

Return the isomorphic function field.

**EXAMPLES:**

```
sage: R.<t> = GF(5)[]
sage: K = R.fraction_field()
sage: K.function_field()
Rational function field in t over Finite Field of size 5
```

See also:

- sage.rings.function_field.RationalFunctionField.field()
**maximal_order()**

Return the maximal order in this fraction field.

**EXAMPLES:**

```python
sage: K = FractionField(GF(5)\['t'\])
sage: K.maximal_order()
Univariate Polynomial Ring in t over Finite Field of size 5
```

**ring_of_integers()**

Return the ring of integers in this fraction field.

**EXAMPLES:**

```python
sage: K = FractionField(GF(5)\['t'\])
sage: K.ring_of_integers()
Univariate Polynomial Ring in t over Finite Field of size 5
```

**class** `sage.rings.fraction_field.FractionField_generic(R, element_class=<class 'sage.rings.fraction_field_element.FractionFieldElement'>, category=Category of quotient fields)

Bases: `Field`

The fraction field of an integral domain.

**base_ring()**

Return the base ring of `self`.

This is the base ring of the ring which this fraction field is the fraction field of.

**EXAMPLES:**

```python
sage: R = Frac(ZZ\['t'\])
sage: R.base_ring()
Integer Ring
```

**characteristic()**

Return the characteristic of this fraction field.

**EXAMPLES:**

```python
sage: R = Frac(ZZ\['t'\])
sage: R.base_ring()
Integer Ring
sage: R.characteristic()
0
sage: R = Frac(GF(5)\['w'\]); R.characteristic()
5
```

**construction()**

**EXAMPLES:**

```python
sage: Frac(ZZ\['x'\]).construction()
(FractionField, Univariate Polynomial Ring in x over Integer Ring)
sage: K = Frac(GF(3)\['t'\])
sage: f, R = K.construction()
```
**gen**(i=0)

Return the i-th generator of self.

EXAMPLES:

```python
sage: R = Frac(PolynomialRing(QQ, 'z', 10)); R
Fraction Field of Multivariate Polynomial Ring in z0, z1, z2, z3, z4, z5, z6, z7, z8, z9 over Rational Field
sage: R.0
z0
sage: R.gen(3)
z3
sage: R.3
z3
```

**is_exact()**

Return if self is exact which is if the underlying ring is exact.

EXAMPLES:

```python
sage: Frac(ZZ['x']).is_exact()
True
sage: Frac(CDF['x']).is_exact()
# nneds sage.rings.complex_double
False
```

**is_field**(proof=True)

Return True, since the fraction field is a field.

EXAMPLES:

```python
sage: Frac(ZZ).is_field()
True
```

**is_finite()**

Tells whether this fraction field is finite.

**Note:** A fraction field is finite if and only if the associated integral domain is finite.

EXAMPLES:

```python
sage: Frac(QQ['a','b','c']).is_finite()
False
```

**ngens()**

This is the same as for the parent object.

EXAMPLES:

5.1. Fraction Field of Integral Domains
sage: R = Frac(PolynomialRing(QQ,'z',10)); R
Fraction Field of Multivariate Polynomial Ring
in z0, z1, z2, z3, z4, z5, z6, z7, z8, z9 over Rational Field
sage: R.ngens()
10

random_element(*args, **kwds)

Return a random element in this fraction field.

The arguments are passed to the random generator of the underlying ring.

EXAMPLES:

sage: F = ZZ['x'].fraction_field()
sage: F.random_element()  # random
(2*x - 8)/(-x^2 + x)

sage: f = F.random_element(degree=5)
sage: f.numerator().degree() == f.denominator().degree()
True
sage: f.denominator().degree() <= 5
True
sage: while f.numerator().degree() != 5:
....:     f = F.random_element(degree=5)

ring()

Return the ring that this is the fraction field of.

EXAMPLES:

sage: R = Frac(QQ['x,y'])
sage: R
Fraction Field of Multivariate Polynomial Ring in x, y over Rational Field
sage: R.ring()
Multivariate Polynomial Ring in x, y over Rational Field

some_elements()

Return some elements in this field.

EXAMPLES:

sage: R.<x> = QQ[]
sage: R.fraction_field().some_elements()
[0, 1, x, 2*x, x/(x^2 + 2*x + 1), 1/x^2, ...
(2*x^2 + 2)/(x^2 + 2*x + 1), (2*x^2 + 2)/x^3, (2*x^2 + 2)/(x^2 - 1), 2]
sage.rings.fraction_field.is_FractionField(x)
Test whether or not \( x \) inherits from \texttt{FractionField_generic}.

**EXAMPLES:**

```python
sage: from sage.rings.fraction_field import is_FractionField
sage: is_FractionField(Frac(ZZ['x']))
True
sage: is_FractionField(QQ)
False
```

## 5.2 Fraction Field Elements

**AUTHORS:**

- William Stein (input from David Joyner, David Kohel, and Joe Wetherell)
- Sebastian Pancratz (2010-01-06): Rewrite of addition, multiplication and derivative to use Henrici’s algorithms [Hor1972]

**class** `sage.rings.fraction_field_element.FractionFieldElement`

**Bases:** `FieldElement`

**EXAMPLES:**

```python
sage: K = FractionField(PolynomialRing(QQ, 'x'))
sage: K
Fraction Field of Univariate Polynomial Ring in x over Rational Field
sage: loads(K.dumps()) == K
True
sage: x = K.gen()
sage: f = (x^3 + x)/(17 - x^19); f
(-x^3 - x)/(x^19 - 17)
sage: loads(f.dumps()) == f
True
```

**denominator()**

Return the denominator of \( \texttt{self} \).

**EXAMPLES:**

```python
sage: R.<x,y> = ZZ[]
sage: f = x/y + 1; f
(x + y)/y
sage: f.denominator()
y
```

**is_one()**

Return True if this element is equal to one.

**EXAMPLES:**

```python
sage: F = ZZ['x,y'].fraction_field()
sage: x,y = F.gens()
```

(continues on next page)
**is_one()**

Return True if this element is equal to one.

**is_zero()**

Return True if this element is equal to zero.

**is_square(root=False)**

Return whether or not self is a perfect square.

If the optional argument root is True, then also returns a square root (or None, if the fraction field element is not square).

**INPUT:**

- root – whether or not to also return a square root (default: False)

**OUTPUT:**

- bool - whether or not a square
- object - (optional) an actual square root if found, and None otherwise.

**EXAMPLES:**

```
sage: R.<t> = QQ[]
sage: (1/t).is_square()
False
sage: (1/t^6).is_square()
True
sage: ((1+t)^4/t^6).is_square()
True
sage: (4*(1+t)^4/t^6).is_square()
True
sage: (2*(1+t)^4/t^6).is_square()
False
sage: ((1+t)/t^6).is_square()
False
sage: (4*(1+t)^4/t^6).is_square(root=True)
(True, (2*t^2 + 4*t + 2)/t^3)
sage: (2*(1+t)^4/t^6).is_square(root=True)
(False, None)
sage: R.<x> = QQ[]
sage: a = 2*(x+1)^2 / (2*(x-1)^2); a
(x^2 + 2*x + 1)/(x^2 - 2*x + 1)
sage: a.is_square()
True
sage: (0/x).is_square()
True
```
General Rings, Ideals, and Morphisms, Release 10.2

```
sage: F = ZZ['x,y'].fraction_field()
sage: x,y = F.gens()
sage: t = F(0)/x
sage: t.is_zero()
True
sage: u = 1/x - 1/x
sage: u.is_zero()
True
sage: u.parent() is F
True
```

**nth_root(n)**

Return a n-th root of this element.

EXAMPLES:

```
sage: R = QQ['t'].fraction_field()
sage: t = R.gen()
sage: p = (t+1)^3 / (t^2+t-1)^3
sage: p.nth_root(3)
(t + 1)/(t^2 + t - 1)
sage: p = (t+1) / (t-1)
sage: p.nth_root(2)
Traceback (most recent call last):
... ValueError: not a 2nd power
```

**numerator()**

Return the numerator of self.

EXAMPLES:

```
sage: R.<x,y> = ZZ[]
sage: f = x/y + 1; f
(x + y)/y
sage: f.numerator()
x + y
```

**reduce()**

Reduce this fraction.

Divides out the gcd of the numerator and denominator. If the denominator becomes a unit, it becomes 1. Additionally, depending on the base ring, the leading coefficients of the numerator and the denominator may be normalized to 1.

Automatically called for exact rings, but because it may be numerically unstable for inexact rings it must be called manually in that case.

EXAMPLES:

```
sage: R.<x> = RealField(10)[]
    # needs sage.rings.real_mpfr
sage: f = (x^2+2*x+1)/(x+1); f
    # needs sage.rings.real_mpfr
```

(continues on next page)
(x^2 + 2.0*x + 1.0)/(x + 1.0)
sage: f.reduce(); f
˓→ needs sage.rings.real_mpfr
x + 1.0

specialization(D=None, phi=None)
Returns the specialization of a fraction element of a polynomial ring

valuation(v=None)
Return the valuation of self, assuming that the numerator and denominator have valuation functions defined on them.

EXAMPLES:

sage: x = PolynomialRing(RationalField(), 'x').gen()
sage: f = (x^3 + x)/(x^2 - 2*x^3)
sage: f
(-1/2*x^2 - 1/2)/(x^2 - 1/2*x)
sage: f.valuation()
-1
sage: f.valuation(x^2 + 1)
1

class sage.rings.fraction_field_element.FractionFieldElement_1poly_field
Bases: FractionFieldElement

A fraction field element where the parent is the fraction field of a univariate polynomial ring over a field.

Many of the functions here are included for coherence with number fields.

is_integral() Returns whether this element is actually a polynomial.

EXAMPLES:

sage: R.<t> = QQ[]
sage: elt = (t^2 + t - 2) / (t + 2); elt # == (t + 2)*(t - 1)/(t + 2)
t - 1
sage: elt.is_integral()
True
sage: elt = (t^2 - t) / (t+2); elt # == t*(t - 1)/(t + 2)
(t^2 - t)/(t + 2)
sage: elt.is_integral()
False

reduce() Pick a normalized representation of self.

In particular, for any a == b, after normalization they will have the same numerator and denominator.

EXAMPLES:

For univariate rational functions over a field, we have:
```python
sage: R.<x> = QQ[]
sage: (2 + 2*x) / (4*x)  # indirect doctest
(1/2*x + 1/2)/x
```

Compare with:

```python
sage: R.<x> = ZZ[]
sage: (2 + 2*x) / (4*x)
(x + 1)/(2*x)
```

**support()**

Returns a sorted list of primes dividing either the numerator or denominator of this element.

**EXAMPLES:**

```python
sage: R.<t> = QQ[]
sage: h = (t^14 + 2*t^12 - 4*t^11 - 8*t^9 + 6*t^8 + 12*t^6 - 4*t^5 - 8*t^3 + t^2 + 2)/(t^6 + 6*t^5 + 9*t^4 - 2*t^2 - 12*t - 18)
sage: h.support()  # needs sage.libs.pari
[t - 1, t + 3, t^2 + 2, t^2 + t + 1, t^4 - 2]
```

```python
sage.rings.fraction_field_element.is_FractionFieldElement(x)
```

Return whether or not `x` is a `FractionFieldElement`.

**EXAMPLES:**

```python
sage: from sage.rings.fraction_field_element import is_FractionFieldElement
sage: R.<x> = ZZ[]
sage: is_FractionFieldElement(x/2)
False
sage: is_FractionFieldElement(2/x)
True
sage: is_FractionFieldElement(1/3)
False
```

```python
sage.rings.fraction_field_element.make_element(parent, numerator, denominator)
```

Used for unpickling `FractionFieldElement` objects (and subclasses).

**EXAMPLES:**

```python
sage: from sage.rings.fraction_field_element import make_element
sage: R = ZZ['x,y']
sage: x,y = R.gens()
sage: F = R.fraction_field()
sage: make_element(F, 1 + x, 1 + y)
(x + 1)/(y + 1)
```

```python
sage.rings.fraction_field_element.make_element_old(parent, cdict)
```

Used for unpickling old `FractionFieldElement` pickles.

**EXAMPLES:**

```python
sage: from sage.rings.fraction_field_element import make_element_old
sage: R.<x,y> = ZZ[]
```

(continues on next page)
sage: F = R.fraction_field()

sage: make_element_old(F, {'_FractionFieldElement__numerator': x + y,
                       '_FractionFieldElement__denominator': x - y})

(x + y)/(x - y)
6.1 Localization

Localization is an important ring construction tool. Whenever you have to extend a given integral domain such that it contains the inverses of a finite set of elements but should allow non injective homomorphic images this construction will be needed. See the example on Ariki-Koike algebras below for such an application.

**EXAMPLES:**

```sage
sage: # needs sage.modules
sage: LZ = Localization(ZZ, (5,11))
sage: m = matrix(LZ, [[5, 7], [0,11]])
sage: m.parent()
Full MatrixSpace of 2 by 2 dense matrices over Integer Ring localized at (5, 11)
sage: ~m  # parent of inverse is different: see documentation of m.__invert__
[ 1/5 -7/55]
[ 0 1/11]
sage: _.parent()
Full MatrixSpace of 2 by 2 dense matrices over Rational Field
sage: mi = matrix(LZ, ~m)
sage: mi.parent()
Full MatrixSpace of 2 by 2 dense matrices over Integer Ring localized at (5, 11)
sage: mi == ~m
True
```

The next example defines the most general ring containing the coefficients of the irreducible representations of the Ariki-Koike algebra corresponding to the three colored permutations on three elements:

```sage
sage: R.<u0, u1, u2, q> = ZZ[]
sage: u = [u0, u1, u2]
sage: S = Set(u)
sage: I = S.cartesian_product(S)
sage: add_units = u + [q, q + 1] + [ui - uj for ui, uj in I if ui != uj]
sage: add_units += [q*ui - uj for ui, uj in I if ui != uj]
sage: L = R.localization(tuple(add_units)); L
Multivariate Polynomial Ring in u0, u1, u2, q over Integer Ring localized at (q, q + 1, u2, u1 - u2, u0 - u1, u2^2*q - u1, u2*q - u0, u1^2*q - u2, u1^2*q - u0, u0^2*q - u2, u0^2*q - u1)
```

Define the representation matrices (of one of the three dimensional irreducible representations):
Check relations of the Ariki-Koike algebra:

```
sage: # needs sage.libs.pari sage.modules
sage: m1*m2*m1*m2 == m2*m1*m2*m1
True
sage: m2*m3*m2 == m3*m2*m3
True
sage: m1*m3 == m3*m1
True
sage: m1**3 - (u0+u1+u2)*m1**2 + (u0*u1+u0*u2+u1*u2)*m1 - u0*u1*u2 == 0
True
sage: m2**2 - (q-1)*m2 - q == 0
True
sage: m3**2 - (q-1)*m3 - q == 0
True
sage: ~m1 in m1.parent()
True
sage: ~m2 in m2.parent()
True
sage: ~m3 in m3.parent()
True
```

Obtain specializations in positive characteristic:

```
sage: # needs sage.libs.pari sage.modules
sage: Fp = GF(17)
sage: f = L.hom((3,5,7,11), codomain=Fp); f
Ring morphism:
  From: Multivariate Polynomial Ring in u0, u1, u2, q over Integer Ring localized at
         (q, q + 1, u2, u1, u1 - u2, u0, u0 - u2, u0 - u1, u2*q - u1, u2*q - u0,
         ul*q - u2, ul*q - u0, u0*q - u2, u0*q - u1)
  To:   Finite Field of size 17
Defn: u0 |--> 3
       u1 |--> 5
       u2 |--> 7
       q |--> 11
sage: mFp1 = matrix({k: f(v) for k, v in m1.dict().items()}); mFp1
[5 0 0]
[0 3 0]
[0 0 3]
sage: mFp1.base_ring()
Finite Field of size 17
(continues on next page)
```
sage: mFp2 = matrix({k: f(v) for k, v in m2.dict().items()}); mFp2
\[
\begin{bmatrix}
  2 & 3 & 0 \\
  9 & 8 & 0 \\
  0 & 0 & 16 \\
\end{bmatrix}
\]

sage: mFp3 = matrix({k: f(v) for k, v in m3.dict().items()}); mFp3
\[
\begin{bmatrix}
  16 & 0 & 0 \\
  0 & 4 & 5 \\
  0 & 7 & 6 \\
\end{bmatrix}
\]

Obtain specializations in characteristic 0:

sage: # needs sage.libs.pari
sage: fQ = L.hom((3,5,7,11), codomain=QQ); fQ
Ring morphism:
   From: Multivariate Polynomial Ring in u0, u1, u2, q over Integer Ring
          localized at (q, q + 1, u2, u1 - u2, u0, u0 - u2, u0 - u1,
             u2*q - u1, u2*q - u0, u1*q - u2, u1*q - u0, u0*q - u2, u0*q - u1)
   To:   Rational Field
      Defn: u0 |--> 3
            u1 |-- 5
            u2 |-- 7
            q |-- 11

sage: # needs sage.libs.pari sage.modules sage.rings.finite_rings
sage: mQ1 = matrix({k: fQ(v) for k, v in m1.dict().items()}); mQ1
\[
\begin{bmatrix}
  5 & 0 & 0 \\
  0 & 3 & 0 \\
  0 & 0 & 3 \\
\end{bmatrix}
\]

sage: mQ1.base_ring()
Rational Field

sage: mQ2 = matrix({k: fQ(v) for k, v in m2.dict().items()}); mQ2
\[
\begin{bmatrix}
  -15 & -14 & 0 \\
  26 & 25 & 0 \\
  0 & 0 & -1 \\
\end{bmatrix}
\]

sage: mQ3 = matrix({k: fQ(v) for k, v in m3.dict().items()}); mQ3
\[
\begin{bmatrix}
  -1 & 0 & 0 \\
  0 & -15/26 & 11/26 \\
  0 & 301/26 & 275/26 \\
\end{bmatrix}
\]

sage: # needs sage.libs.pari sage.libs.singular
sage: S.<x, y, z, t> = QQ[]
sage: T = S.quo(x + y + z)
sage: F = T.fraction_field()
sage: fF = L.hom((x, y, z, t), codomain=F); fF
Ring morphism:
   From: Multivariate Polynomial Ring in u0, u1, u2, q over Integer Ring
          localized at (q, q + 1, u2, u1 - u2, u0, u0 - u2, u0 - u1,
             u2*q - u1, u2*q - u0, u1*q - u2, u1*q - u0, u0*q - u2, u0*q - u1)
   To:   Fraction Field of Quotient of Multivariate Polynomial Ring in x, y, z, t
          over Rational Field by the ideal (x + y + z)
      Defn: u0 |--> -ybar - zbar
            u1 |--> ybar

(continues on next page)
The localization generalizes the construction of the field of fractions of an integral domain to an arbitrary ring. Given a (not necessarily commutative) ring $R$ and a subset $S$ of $R$, there exists a ring $R[S^{-1}]$ together with the ring homomorphism $R \to R[S^{-1}]$ that “inverts” $S$; that is, the homomorphism maps elements in $S$ to unit elements in $R[S^{-1}]$ and, moreover, any ring homomorphism from $R$ that “inverts” $S$ uniquely factors through $R[S^{-1}]$.

The ring $R[S^{-1}]$ is called the localization of $R$ with respect to $S$. For example, if $R$ is a commutative ring and $f$ an element in $R$, then the localization consists of elements of the form $r/f, r \in R, n \geq 0$ (to be precise, $R[f^{-1}] = R[t]/(ft - 1)$).

The above text is taken from Wikipedia. The construction here used for this class relies on the construction of the field of fraction and is therefore restricted to integral domains.

Accordingly, this class is inherited from `IntegralDomain` and can only be used in that context. Furthermore, the base ring should support `sage.structure.element.CommutativeRingElement.divides()` and the exact division operator `// (sage.structure.element.Element.__floordiv__)` in order to guarantee an successful application.

**INPUT:**

- `base_ring` – an instance of `Ring` allowing the construction of `fraction_field()` (that is an integral domain)
- `extra_units` – tuple of elements of `base_ring` which should be turned into units
- `names` – passed to `IntegralDomain`
- `normalize` – (optional, default: True) passed to `IntegralDomain`
- `category` – (optional, default: None) passed to `IntegralDomain`
- `warning` – (optional, default: True) to suppress a warning which is thrown if self cannot be represented uniquely

**REFERENCES:**

- Wikipedia article `Ring_(mathematics)#Localization`

**EXAMPLES:**
sage: L = Localization(ZZ, (3, 5))
sage: 1/45 in L
    True
sage: 1/43 in L
    False

sage: Localization(L, (7, 11))
    Integer Ring localized at (3, 5, 7, 11)

sage: _.is_subring(QQ)
    True
sage: L(~7)
    Traceback (most recent call last):
      ... ValueError: factor 7 of denominator is not a unit

sage: Localization(Zp(7), (3, 5))
    # needs sage.rings.padics
    Traceback (most recent call last):
      ... ValueError: all given elements are invertible in 7-adic Ring with capped relative precision 20

sage: # needs sage.libs.pari
sage: R.<x> = ZZ[]
sage: L = R.localization(x**2 + 1)
sage: s = (x+5)/(x**2+1)
sage: s in L
    True
sage: t = (x+5)/(x**2+2)
sage: t in L
    False
sage: L(t)
    Traceback (most recent call last):
      ... TypeError: fraction must have unit denominator
sage: L(s) in R
    False
sage: y = L(x)
sage: g = L(s)
sage: g.parent()
    Univariate Polynomial Ring in x over Integer Ring localized at (x^2 + 1,)
sage: f = (y+5)/(y**2+1); f
    (x + 5)/(x^2 + 1)

sage: f == g
    True

sage: (y+5)/(y**2+2)
    Traceback (most recent call last):
      ... ValueError: factor x^2 + 2 of denominator is not a unit

sage: Lau.<u, v> = LaurentPolynomialRing(ZZ)
    # needs sage.modules

(continues on next page)
More examples will be shown typing `sage.rings.localization`?

**Element**

alias of `LocalizationElement`

**characteristic()**

Return the characteristic of `self`.

**EXAMPLES:**

```
sage: # needs sage.libs.pari
sage: R.<a> = GF(5)[]
sage: L = R.localization((a**2 - 3, a))
sage: L.characteristic()
sage: 5
```

**fraction_field()**

Return the fraction field of `self`.

**EXAMPLES:**

```
sage: # needs sage.libs.pari
sage: R.<a> = GF(5)[]
sage: L = Localization(R, (a**2 - 3, a))
sage: L.fraction_field()  # Fraction Field of Univariate Polynomial Ring in a over Finite Field of size 5
Fraction Field of Univariate Polynomial Ring in a over Finite Field of size 5
sage: L.is_subring(_)
sage: True
```

**gen(i)**

Return the `i`-th generator of `self` which is the `i`-th generator of the base ring.

**EXAMPLES:**

```
sage: R.<x, y> = ZZ[]
sage: R.localization((x**2 + 1, y - 1)).gen(0)  # needs sage.libs.pari
sage: x
sage: ZZ.localization(2).gen(0)
sage: 1
```

**gens()**

Return a tuple whose entries are the generators for this object, in order.

**EXAMPLES:**
```python
sage: R.<x, y> = ZZ[]
sage: Localization(R, (x^2 + 1, y - 1)).gens()  # needs sage.libs.pari
(x, y)
sage: Localization(ZZ, 2).gens()
(1,)
```

**is_field(proof=True)**

Return True if this ring is a field.

**INPUT:**

- proof – (default: True) Determines what to do in unknown cases

**ALGORITHM:**

If the parameter proof is set to True, the returned value is correct but the method might throw an error. Otherwise, if it is set to False, the method returns True if it can establish that self is a field and False otherwise.

**EXAMPLES:**

```python
sage: R = ZZ.localization((2, 3))
sage: R.is_field()
False
```

**krull_dimension()**

Return the Krull dimension of this localization.

Since the current implementation just allows integral domains as base ring and localization at a finite set of elements the spectrum of self is open in the irreducible spectrum of its base ring. Therefore, by density we may take the dimension from there.

**EXAMPLES:**

```python
sage: R = ZZ.localization((2, 3))
sage: R.krull_dimension()
1
```

**ngens()**

Return the number of generators of self according to the same method for the base ring.

**EXAMPLES:**

```python
sage: R.<x, y> = ZZ[]
sage: Localization(R, (x^2 + 1, y - 1)).ngens()  # needs sage.libs.pari
2
sage: Localization(ZZ, 2).ngens()
1
```

class sage.rings.localization.LocalizationElement(parent, x)

Bases: IntegralDomainElement

Element class for localizations of integral domains
INPUT:

• **parent** – instance of *Localization*

• **x** – instance of *FractionFieldElement* whose parent is the fraction field of the parent's base ring

EXAMPLES:

```
sage: # needs sage.libs.pari
sage: from sage.rings.localization import LocalizationElement
sage: P.<x,y,z> = GF(5)[]
sage: L = P.localization((x, y*z - x))

sage: LocalizationElement(L, 4/(y*z-x)**2)
(-1)/(y^2*z^2 - 2*x*y*z + x^2)

sage: _.parent()
Multivariate Polynomial Ring in x, y, z over Finite Field of size 5 localized at (x, y*z - x)
```

denominator()

Return the denominator of *self*.

EXAMPLES:

```
sage: L = Localization(ZZ, (3,5))

sage: L(7/15).denominator()
15
```

factor(*proof=None*)

Return the factorization of this polynomial.

INPUT:

• **proof** – (optional) if given it is passed to the corresponding method of the numerator of *self*

EXAMPLES:

```
sage: P.<X, Y> = QQ[

sage: L = P.localization(X - Y)

sage: x, y = L.gens()

sage: p = (x^2 - y^2)/(x-y)^2

# needs sage.libs.singular

sage: p.factor()
# needs sage.libs.singular

(1/(x - y)) * (x + y)
```

inverse_of_unit()

Return the inverse of *self*.

EXAMPLES:

```
sage: P.<x,y,z> = ZZ[]

sage: L = Localization(P, x*y*z)

sage: L(x*y*z).inverse_of_unit()
# needs sage.libs.singular

1/(x*y*z)

sage: L(z).inverse_of_unit()
# needs sage.libs.singular
```

(continues on next page)
### is_unit()

Return True if self is a unit.

**EXAMPLES:**

```python
sage: # needs sage.libs.pari sage.singular
sage: P.<x,y,z> = QQ[]
sage: L = P.localization((x, y^*z))
sage: L(y^*z).is_unit()  # True
sage: L(z).is_unit()  # True
sage: L(x*y*z).is_unit()  # True
```

### numerator()

Return the numerator of self.

**EXAMPLES:**

```python
sage: L = ZZ.localization((3,5))
sage: L(7/15).numerator()  # 7
```

### sage.rings.localization.normalize_extra_units(base_ring, add_units, warning=True)

Function to normalize input data.

The given list will be replaced by a list of the involved prime factors (if possible).

**INPUT:**

- base_ring – an instance of `IntegralDomain`
- add_units – list of elements from base ring
- warning – (optional, default: True) to suppress a warning which is thrown if no normalization was possible

**OUTPUT:**

List of all prime factors of the elements of the given list.

**EXAMPLES:**

```python
sage: from sage.rings.localization import normalize_extra_units
sage: normalize_extra_units(ZZ, [3, -15, 45, 9, 2, 50])
[2, 3, 5]

sage: P.<x,y,z> = ZZ[]

sage: normalize_extra_units(P, # needs sage.libs.pari
                        # list of elements from base ring
                        [3^x, z^y^z^2, 2^z, 18^x^y^z^2, x^z, 6^x^z^z, 5])
[2, 3, 5, z, y, x]
```

(continues on next page)
....:  [3*x, z*y**2, 2*z, 18*(x*y*z)**2, x*z, 6*x*z, 5])
[z, y, x]
sage: # needs sage.libs.singular
sage: R.<x, y> = ZZ[]
sage: Q.<a, b> = R.quo(x**2 - 5)
sage: p = b**2 - 5
sage: p == (b-a)*(b+a)
True
sage: normalize_extra_units(Q, [p])
#...
doctest:...: UserWarning: Localization may not be represented uniquely
[b^2 - 5]
sage: normalize_extra_units(Q, [p], warning=False)
#...

needs sage.libs.pari
[116 Chapter 6. Localization
7.1 Extension of rings

Sage offers the possibility to work with ring extensions $L/K$ as actual parents and perform meaningful operations on them and their elements.

The simplest way to build an extension is to use the method `sage.categories.commutative_rings.CommutativeRings.ParentMethods.over()` on the top ring, that is $L$. For example, the following line constructs the extension of finite fields $\mathbb{F}_{5^4}/\mathbb{F}_{5^2}$:

```sage
sage: GF(5^4).over(GF(5^2))
```

By default, Sage reuses the canonical generator of the top ring (here $z_4 \in \mathbb{F}_{5^4}$), together with its name. However, the user can customize them by passing in appropriate arguments:

```sage
sage: # needs sage.rings.finite_rings
sage: F = GF(5^2)
sage: k = GF(5^4)
sage: z4 = k.gen()
sage: K.<a> = k.over(F, gen=1-z4); K
```

The base of the extension is available via the method `base()` (or equivalently `base_ring()`):

```sage
sage: K.base()
```

It is also possible to build an extension on top of another extension, obtaining this way a tower of extensions:

```sage
sage: L.<b> = GF(5^8).over(K); L
```

The base of the extension is available via the method `base()` (or equivalently `base_ring()`):

```sage
sage: L.base()
```

Finite Field in $z_2$ of size 5^2
The method `bases()` gives access to the complete list of rings in a tower:

```python
sage: L.bases()  # needs sage.rings.finite_rings
[Field in b with defining polynomial x^2 + (4*z2 + 3*a)*x + 1 - a over its base,
Field in a with defining polynomial x^2 + z2*x + 4 over its base,
Finite Field in z2 of size 5^2]
```

Once we have constructed an extension (or a tower of extensions), we have interesting methods attached to it. As a basic example, one can compute a basis of the top ring over any base in the tower:

```python
sage: L.basis_over(K)  # needs sage.rings.finite_rings
[1, b]
sage: L.basis_over(F)  # needs sage.rings.finite_rings
[1, a, b, a*b]
```

When the base is omitted, the default is the natural base of the extension:

```python
sage: L.basis_over()  # needs sage.rings.finite_rings
[1, b]
```

The method `sage.rings.ring_extension_element.RingExtensionWithBasis.vector()` computes the coordinates of an element according to the above basis:

```python
sage: u = a + 2*b + 3*a*b  # needs sage.rings.finite_rings
sage: u.vector()  # over K
(a, 2 + 3*a)
sage: u.vector(F)  # needs sage.rings.finite_rings
(0, 1, 2, 3)
```

One can also compute traces and norms with respect to any base of the tower:

```python
sage: u.trace()  # over K
(2*z2 + 1) + (2*z2 + 1)*a
sage: u.trace(F)  # over K, then over F
z2 + 1
sage: u.norm()  # over K
(z2 + 1) + (4*z2 + 2)*a
sage: u.norm(F)  # needs sage.rings.finite_rings
2*z2 + 2
```

And minimal polynomials:

```python
sage: u.minpoly()  # needs sage.rings.finite_rings
```

(continues on next page)
AUTHOR:

- Xavier Caruso (2019)

class sage.rings.ring_extension.RingExtensionFactory

    Bases: UniqueFactory

    Factory for ring extensions.

    create_key_and_extra_args(ring, defining_morphism=None, gens=None, names=None,
                                constructors=None)

    Create a key and return it together with a list of constructors of the object.

    INPUT:

    - ring -- a commutative ring
    - defining_morphism -- a ring homomorphism or a commutative ring or None (default: None); the
      defining morphism of this extension or its base (if it coerces to ring)
    - gens -- a list of generators of this extension (over its base) or None (default: None);
    - names -- a list or a tuple of variable names or None (default: None)
    - constructors -- a list of constructors; each constructor is a pair (class, arguments) where class is
      the class implementing the extension and arguments is the dictionary of arguments to pass in to init
      function

    create_object(version, key, **extra_args)

    Return the object associated to a given key.

class sage.rings.ring_extension.RingExtensionFractionField

    Bases: RingExtension_generic

    A class for ring extensions of the form `\text{Frac}(A)/A`.

    Element

    alias of RingExtensionFractionFieldElement

    ring()

    Return the ring whose fraction field is this extension.

    EXAMPLES:

    sage: # needs sage.rings.number_field
    sage: x = polygen(ZZ, 'x')
    sage: A.<a> = ZZ.extension(x^2 - 2)
    sage: OK = A.over()
    sage: K = OK.fraction_field(); K
    Fraction Field of
      Order in Number Field in a with defining polynomial x^2 - 2 over its base
    sage: K.ring()
    Order in Number Field in a with defining polynomial x^2 - 2 over its base

7.1. Extension of rings
class sage.rings.ring_extension.RingExtensionWithBasis

Bases: RingExtension_generic

A class for finite free ring extensions equipped with a basis.

Element

alias of RingExtensionWithBasisElement

basis_over(base=None)

Return a basis of this extension over base.

INPUT:

- base – a commutative ring (which might be itself an extension)

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: F.<a> = GF(5^2).over() # over GF(5)
sage: K.<b> = GF(5^4).over(F)
sage: L.<c> = GF(5^12).over(K)
sage: L.basis_over(K)
[1, c, c^2]
sage: L.basis_over(F)
[1, b, c, b*c, c^2, b*c^2]
sage: L.basis_over(GF(5))
[1, a, b, a*b, c, a*c, b*c, a*b*c, c^2, a*c^2, b*c^2, a*b*c^2]
```

If base is omitted, it is set to its default which is the base of the extension:

```
sage: L.basis_over()               # needs sage.rings.finite_rings
[1, c, c^2]
sage: K.basis_over()              # needs sage.rings.finite_rings
[1, b]
```

Note that base must be an explicit base over which the extension has been defined (as listed by the method bases()):

```
sage: L.degree_over(GF(5^6))       # needs sage.rings.finite_rings
Traceback (most recent call last):
...
ValueError: not (explicitly) defined over Finite Field in z6 of size 5^6
```

fraction_field(extend_base=False)

Return the fraction field of this extension.

INPUT:

- extend_base – a boolean (default: False):
If `extend_base` is `False`, the fraction field of the extension $L/K$ is defined as $\text{Frac}(L)/L/K$, except is $L$ is already a field in which base the fraction field of $L/K$ is $L/K$ itself.

If `extend_base` is `True`, the fraction field of the extension $L/K$ is defined as $\text{Frac}(L)/\text{Frac}(K)$ (provided that the defining morphism extends to the fraction fields, i.e. is injective).

EXAMPLES:

```python
sage: # needs sage.rings.number_field
sage: x = polygen(ZZ, 'x')
sage: A.<a> = ZZ.extension(x^2 - 5)
sage: OK = A.over()  # over ZZ
sage: OK
Order in Number Field in a with defining polynomial x^2 - 5 over its base
sage: K1 = OK.fraction_field(); K1
Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5 over its base
sage: K1.bases()
[Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5 over its base,
 Order in Number Field in a with defining polynomial x^2 - 5 over its base,
 Integer Ring]
sage: K2 = OK.fraction_field(extend_base=True); K2
Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5 over its base
sage: K2.bases()
[Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5 over its base,
 Rational Field]
```

Note that there is no coercion map between $K_1$ and $K_2$:

```python
sage: K1.has_coerce_map_from(K2)  # needs sage.rings.number_field
False
sage: K2.has_coerce_map_from(K1)  # needs sage.rings.number_field
False
```

We check that when the extension is a field, its fraction field does not change:

```python
sage: K1.fraction_field() is K1  # needs sage.rings.number_field
True
sage: K2.fraction_field() is K2  # needs sage.rings.number_field
True
```

**free_module** *(base=None, map=True)*

Return a free module $V$ over `base` which is isomorphic to this ring

**INPUT:**

- `base` – a commutative ring (which might be itself an extension) or `None` (default: `None`)
- `map` – boolean (default `True`); whether to return isomorphisms between this ring and $V
OUTPUT:

• A finite-rank free module \( V \) over base
• The isomorphism from \( V \) to this ring corresponding to the basis output by the method \( \text{basis}_\text{over}() \) (only included if \( \text{map} \) is True)
• The reverse isomorphism of the isomorphism above (only included if \( \text{map} \) is True)

EXAMPLES:

```sage
sage: F = GF(11)
sage: K.<a> = GF(11^2).over()
# needs sage.rings.finite_rings
sage: L.<b> = GF(11^6).over(K)
# needs sage.rings.finite_rings
```

Forgetting a part of the multiplicative structure, the field \( L \) can be viewed as a vector space of dimension 3 over \( K \), equipped with a distinguished basis, namely \((1, b, b^2)\):

```sage
sage: # needs sage.rings.finite_rings
sage: V, i, j = L.free_module(K)
sage: V
Vector space of dimension 3 over Field in a with defining polynomial \( x^2 + 7*x + 2 \) over its base
sage: i
Generic map:
From: Vector space of dimension 3 over Field in a with defining polynomial \( x^2 + 7*x + 2 \) over its base
To: Field in b with defining polynomial \( x^3 + (7 + 2*a)*x^2 + (2 - a)*x - a \) over its base
sage: j
Generic map:
From: Field in b with defining polynomial \( x^3 + (7 + 2*a)*x^2 + (2 - a)*x - a \) over its base
To: Vector space of dimension 3 over Field in a with defining polynomial \( x^2 + 7*x + 2 \) over its base
sage: j(b)
(0, 1, 0)
sage: i((1, a, a+1))
1 + a*b + (1 + a)*b^2
```

Similarly, one can view \( L \) as a \( F \)-vector space of dimension 6:

```sage
sage: # needs sage.rings.finite_rings
sage: V, i, j, = L.free_module(F)
sage: V
Vector space of dimension 6 over Finite Field of size 11
```

In this case, the isomorphisms between \( V \) and \( L \) are given by the basis \((1, a, b, ab, b^2, ab^2)\):

```sage
sage: j(a*b) # needs sage.rings.finite_rings (0, 0, 0, 1, 0, 0)
sage: i((1,2,3,4,5,6)) # needs sage.rings.finite_rings (1 + 2*a) + (3 + 4*a)*b + (5 + 6*a)*b^2
```

When base is omitted, the default is the base of this extension:
Vector space of dimension 3 over Field in a with defining polynomial \( x^2 + 7x + 2 \) over its base

Note that base must be an explicit base over which the extension has been defined (as listed by the method bases():)

```
sage: L.degree(GF(11^3))  # needs sage.rings.finite_rings
Traceback (most recent call last):
...
ValueError: not (explicitly) defined over Finite Field in z3 of size 11^3
```

**class** *sage.rings.ring_extension.RingExtensionWithGen*

**Bases:** *RingExtensionWithBasis*

A class for finite free ring extensions generated by a single element

**fraction_field(**extend_base=False**)**

Return the fraction field of this extension.

**INPUT:**

- **extend_base** — a boolean (default: False);

If **extend_base** is False, the fraction field of the extension \( L/K \) is defined as \( \text{Frac}(L)/L/K \), except if \( L \) is already a field in which base the fraction field of \( L/K \) is \( L/K \) itself.

If **extend_base** is True, the fraction field of the extension \( L/K \) is defined as \( \text{Frac}(L)/\text{Frac}(K) \) (provided that the defining morphism extends to the fraction fields, i.e. is injective).

**EXAMPLES:**

```
sage: # needs sage.rings.number_field
sage: x = polygen(ZZ, 'x')
sage: A.<a> = ZZ.extension(x^2 - 5)
sage: OK = A.over()  # over ZZ
sage: OK
Order in Number Field in a with defining polynomial x^2 - 5 over its base
sage: K1 = OK.fraction_field(); K1
Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5 over its base
sage: K1.bases()
[Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5 over its base, Integer Ring]
sage: K2 = OK.fraction_field(extend_base=True); K2
Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5 over its base
sage: K2.bases()
[Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5 over its base, Rational Field]
```

Note that there is no coercion map between \( K_1 \) and \( K_2 \):
We check that when the extension is a field, its fraction field does not change:

```
sage: K1.fraction_field() is K1  # needs sage.rings.number_field
True
sage: K2.fraction_field() is K2  # needs sage.rings.number_field
True
```

gens(base=None)

Return the generators of this extension over \( \text{base} \).

INPUT:

- \( \text{base} \) – a commutative ring (which might be itself an extension) or \text{None} (default: \text{None})

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<a> = GF(5^2).over()  # over GF(5)
sage: K.gens()
(a,)
sage: L.<b> = GF(5^4).over(K)
sage: L.gens()
(b,)
sage: L.gens(GF(5))
(b, a)
```

modulus(var=’x’)

Return the defining polynomial of this extension, that is the minimal polynomial of the given generator of this extension.

INPUT:

- \( \text{var} \) – a variable name (default: \( x \))

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<u> = GF(7^10).over(GF(7^2)); K
Field in u with defining polynomial x^5 + (6*z2 + 4)*x^4 + (3*z2 + 5)*x^3 + (2*z2 + 2)*x^2 + 4*x + 6*z2 over its base
sage: P = K.modulus(); P
x^5 + (6*z2 + 4)*x^4 + (3*z2 + 5)*x^3 + (2*z2 + 2)*x^2 + 4*x + 6*z2
sage: P(u)
0
```

We can use a different variable name:
```python
sage: K.modulus('y')
needs sage.rings.finite_rings
y^5 + (6*z2 + 4)*y^4 + (3*z2 + 5)*y^3 + (2*z2 + 2)*y^2 + 4*y + 6*z2
```

### class sage.rings.ring_extension.RingExtension_generic

Bases: `CommutativeAlgebra`

A generic class for all ring extensions.

#### Element

alias of `RingExtensionElement`

##### absolute_base()

Return the absolute base of this extension.

By definition, the absolute base of an iterated extension $K_n/\cdots K_2/K_1$ is the ring $K_1$.

**EXAMPLES:**

```python
sage: F = GF(5^2).over()
# over GF(5)
sage: K = GF(5^4).over(F)
sage: L = GF(5^12).over(K)
sage: F.absolute_base()
Finite Field of size 5
sage: K.absolute_base()
Finite Field of size 5
sage: L.absolute_base()
Finite Field of size 5
```

See also:

`base()`, `bases()`, `is_defined_over()`

##### absolute_degree()

Return the degree of this extension over its absolute base

**EXAMPLES:**

```python
sage: A = GF(5^4).over(GF(5^2))
sage: B = GF(5^12).over(A)
sage: A.absolute_degree()
2
sage: B.absolute_degree()
6
```

See also:

`degree()`, `relative_degree()`

##### backend(force=False)

Return the backend of this extension.

**INPUT:**

- `force` – a boolean (default: `False`); if `False`, raise an error if the backend is not exposed

**EXAMPLES:**

7.1. Extension of rings
sage: # needs sage.rings.finite_rings
sage: K = GF(5^3)
sage: E = K.over()
sage: E
Field in z3 with defining polynomial x^3 + 3*x + 3 over its base
sage: E.backend()
Finite Field in z3 of size 5^3
sage: E.backend() is K
True

base()
Return the base of this extension.

EXAMPLES:

sage: F = GF(5^2)
# needs sage.rings.finite_rings
sage: K = GF(5^4).over(F)
# needs sage.rings.finite_rings
sage: K.base()
# needs sage.rings.finite_rings
Finite Field in z2 of size 5^2

In case of iterated extensions, the base is itself an extension:

sage: L = GF(5^8).over(K)
# needs sage.rings.finite_rings
sage: L.base()
# needs sage.rings.finite_rings
Field in z4 with defining polynomial x^2 + (3*z2 + 3)*x + z2 over its base
sage: L.base() is K
# needs sage.rings.finite_rings
True

See also:
bases(), absolute_base(), is_defined_over()

bases()
Return the list of successive bases of this extension (including itself).

EXAMPLES:

sage: # needs sage.rings.finite_rings
sage: F = GF(5^2).over() # over GF(5)
sage: K = GF(5^4).over(F)
sage: L = GF(5^12).over(K)
sage: F.bases()
[Field in z2 with defining polynomial x^2 + 4*x + 2 over its base,
 Finite Field of size 5]
sage: K.bases()
[Field in z4 with defining polynomial x^2 + (3 - z2)*x + z2 over its base,
 Field in z2 with defining polynomial x^2 + 4*x + 2 over its base,
 Finite Field of size 5]
sage: L.bases()
(continues on next page)
| Field in z12 with defining polynomial |
| x^3 + (1 + (2 - z2)*z4)*x^2 + (2 + 2*z4)*x - z4 over its base, |
| Field in z4 with defining polynomial x^2 + (3 - z2)*x + z2 over its base, |
| Field in z2 with defining polynomial x^2 + 4*x + 2 over its base, |
| Finite Field of size 5 |

See also:

`base()`, `absolute_base()`, `is_defined_over()`

**characteristic()**

Return the characteristic of the extension as a ring.

**OUTPUT:**

A prime number or zero.

**EXAMPLES:**

```sage
sage: # needs sage.rings.finite_rings
sage: F = GF(5^2).over()  # over GF(5)
sage: K = GF(5^4).over(F)
sage: L = GF(5^12).over(K)
sage: F.characteristic()
5
sage: K.characteristic()
5
sage: L.characteristic()
5

sage: F = RR.over(ZZ)
sage: F.characteristic()
0

sage: F = GF(11)
sage: A.<x> = F[]
sage: K = Frac(F).over(F)
sage: K.characteristic()
11

sage: E = GF(7).over(ZZ)
sage: E.characteristic()
7
```

**construction()**

Return the functorial construction of this extension, if defined.

**EXAMPLES:**

```sage
sage: E = GF(5^3).over()
# needs sage.rings.finite_rings
sage: E.constructor()
... # needs sage.rings.finite_rings
```
defining_morphism(base=None)

Return the defining morphism of this extension over base.

INPUT:

• base – a commutative ring (which might be itself an extension) or None (default: None)

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: F = GF(5^2)
sage: K = GF(5^4).over(F)
sage: L = GF(5^12).over(K)
sage: K.defining_morphism()
Ring morphism:
  From: Finite Field in z2 of size 5^2
  To: Field in z4 with defining polynomial x^2 + (4*z2 + 3)*x + z2 over its
    base
  Defn: z2 |--> z2
sage: L.defining_morphism()
Ring morphism:
  From: Field in z4 with defining polynomial x^2 + (4*z2 + 3)*x + z2 over its
    base
  To: Field in z12 with defining polynomial
    x^3 + (1 + (4*z2 + 2)*z4)*x^2 + (2 + 2*z4)*x - z4 over its base
  Defn: z4 |--> z4
```

One can also pass in a base over which the extension is explicitly defined (see also is_defined_over()):

```
sage: L.defining_morphism(F)  # needs sage.rings.finite_rings
Ring morphism:
  From: Finite Field in z2 of size 5^2
  To: Field in z12 with defining polynomial
    x^3 + (1 + (4*z2 + 2)*z4)*x^2 + (2 + 2*z4)*x - z4 over its base
  Defn: z2 |--> z2
sage: L.defining_morphism(GF(5))  # needs sage.rings.finite_rings
Traceback (most recent call last):
  ... 
ValueError: not (explicitly) defined over Finite Field of size 5
```

degree(base)

Return the degree of this extension over base.

INPUT:

• base – a commutative ring (which might be itself an extension)

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: A = GF(5^4).over(GF(5^2))
sage: B = GF(5^12).over(A)
sage: A.degree(GF(5^2))
2
```
Note that base must be an explicit base over which the extension has been defined (as listed by the method `bases()`):

```python
sage: A.degree(GF(5))  # needs sage.rings.finite_rings
Traceback (most recent call last):
  ... ValueError: not (explicitly) defined over Finite Field of size 5
```

See also:

`relative_degree()`, `absolute_degree()`

### degree_over(base=None)

Return the degree of this extension over base.

**INPUT:**

- base – a commutative ring (which might be itself an extension) or `None` (default: `None`)

**EXAMPLES:**

```python
sage: # needs sage.rings.finite_rings
sage: F = GF(5^2)
sage: K = GF(5^4).over(F)
sage: L = GF(5^12).over(K)
sage: K.degree_over(F) 2
sage: L.degree_over(K) 3
sage: L.degree_over(F) 6
```

If base is omitted, the degree is computed over the base of the extension:

```python
sage: K.degree_over()  # needs sage.rings.finite_rings
2
sage: L.degree_over()  # needs sage.rings.finite_rings
3
```

Note that base must be an explicit base over which the extension has been defined (as listed by the method `bases()`):

```python
sage: K.degree_over(GF(5))  # needs sage.rings.finite_rings
Traceback (most recent call last):
  ...
ValueError: not (explicitly) defined over Finite Field of size 5
```
fraction_field(extend_base=False)

Return the fraction field of this extension.

INPUT:

• extend_base – a boolean (default: False):

If extend_base is False, the fraction field of the extension $L/K$ is defined as $\text{Frac}(L)/L/K$, except if $L$ is already a field in which base the fraction field of $L/K$ is $L/K$ itself.

If extend_base is True, the fraction field of the extension $L/K$ is defined as $\text{Frac}(L)/\text{Frac}(K)$ (provided that the defining morphism extends to the fraction fields, i.e. is injective).

EXAMPLES:

```
sage: # needs sage.rings.number_field
sage: x = polygen(ZZ, 'x')
sage: A.<a> = ZZ.extension(x^2 - 5)
sage: OK = A.over()  # over ZZ
sage: OK
Order in Number Field in a with defining polynomial x^2 - 5 over its base
sage: K1 = OK.fraction_field(); K1
Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5 over its base
sage: K1.bases()
[Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5 over its base, Order in Number Field in a with defining polynomial x^2 - 5 over its base, Integer Ring]
sage: K2 = OK.fraction_field(extend_base=True); K2
Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5 over its base
sage: K2.bases()
[Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5 over its base, Rational Field]
```

Note that there is no coercion between $K_1$ and $K_2$:

```
sage: K1.has_coerce_map_from(K2)  # needs sage.rings.number_field
False
sage: K2.has_coerce_map_from(K1)  # needs sage.rings.number_field
False
```

We check that when the extension is a field, its fraction field does not change:

```
sage: K1.fraction_field() is K1  # needs sage.rings.number_field
True
sage: K2.fraction_field() is K2  # needs sage.rings.number_field
True
```

from_base_ring(r)

Return the canonical embedding of r into this extension.
INPUT:

• r – an element of the base of the ring of this extension

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: k = GF(5)
sage: K.<u> = GF(5^2).over(k)
sage: L.<v> = GF(5^4).over(K)
sage: x = L.from_base_ring(k(2)); x
  2
sage: x.parent()  #Field in v with defining polynomial x^2 + (3 - u)*x + u over its base
sage: x = L.from_base_ring(u); x
  u
sage: x.parent()  #Field in v with defining polynomial x^2 + (3 - u)*x + u over its base
```

`gen()`

Return the first generator of this extension.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K = GF(5^2).over()  # over GF(5)
                        #  ⎯⎯ needs sage.rings.finite_rings
sage: x = K.gen(); x      #  ⎯⎯ needs sage.rings.finite_rings
z2
```

Observe that the generator lives in the extension:

```
sage: x.parent()  #Field in z2 with defining polynomial x^2 + 4*x + 2 over its base
                        #  ⎯⎯ needs sage.rings.finite_rings
sage: x.parent()  #is K  #  ⎯⎯ needs sage.rings.finite_rings
True
```

`gens(base=None)`

Return the generators of this extension over base.

INPUT:

• base – a commutative ring (which might be itself an extension) or None (default: None); if omitted, use the base of this extension

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<a> = GF(5^2).over()  # over GF(5)
                        #  ⎯⎯ needs sage.rings.finite_rings
sage: K.gens()  #  ⎯⎯ needs sage.rings.finite_rings
(a,
```

(continues on next page)
sage: L.gens(GF(5))
(b, a)
sage: S.<x> = QQ[]
sage: T.<y> = S[]
sage: T.over(S).gens()
(y,)
sage: T.over(QQ).gens()
(y, x)

\texttt{hom} (\texttt{im\_gens=\texttt{None}}, \texttt{codomain=\texttt{None}}, \texttt{base\_map=\texttt{None}}, \texttt{category=\texttt{None}}, \texttt{check=\texttt{True}})

Return the unique homomorphism from this extension to \texttt{codomain} that sends \texttt{self.gens()} to the entries of \texttt{im\_gens} and induces the map \texttt{base\_map} on the base ring.

\textbf{INPUT:}

- \texttt{im\_gens} – the images of the generators of this extension
- \texttt{codomain} – the codomain of the homomorphism; if omitted, it is set to the smallest parent containing all the entries of \texttt{im\_gens}
- \texttt{base\_map} – a map from one of the bases of this extension into something that coerces into the codomain; if omitted, coercion maps are used
- \texttt{category} – the category of the resulting morphism
- \texttt{check} – a boolean (default: True); whether to verify that the images of generators extend to define a map (using only canonical coercions)

\textbf{EXAMPLES:}

sage: K.<a> = GF(5^2).over()
# over GF(5)
˓→ needs sage.rings.finite_rings
sage: L.<b> = GF(5^6).over(K)
# over GF(5^6)
˓→ needs sage.rings.finite_rings

We define (by hand) the relative Frobenius endomorphism of the extension \(L/K\):

sage: L.hom([b^25])  # \texttt{needs sage.rings.finite_rings}
Ring endomorphism of
Field in b with defining polynomial x^3 + (2 + 2*a)*x - a over its base
Defn: b |--> 2 + 2*a*b + (2 - a)*b^2

Defining the absolute Frobenius of \(L\) is a bit more complicated because it is not a homomorphism of \(K\)-algebras. For this reason, the construction \(L.\text{hom}([b^5])\) fails:

sage: L.hom([b^5])  # \texttt{needs sage.rings.finite_rings}
Traceback (most recent call last):
...
ValueError: images do not define a valid homomorphism

What we need is to specify a base map:
As a shortcut, we may use the following construction:

```python
sage: phi = L.hom([b^5, a^5]); phi  # needs sage.rings.finite_rings
Ring endomorphism of
Field in b with defining polynomial x^3 + (2 + 2*a)*x - a over its base
Defn: b |--> (-1 + a) + (1 + 2*a)*b + a*b^2
    with map on base ring:
      a |--> 1 - a
```

is_defined_over(base)

Return whether or not base is one of the bases of this extension.

INPUT:

- base -- a commutative ring, which might be itself an extension

EXAMPLES:

```python
sage: # needs sage.rings.finite_rings
sage: A = GF(5^4).over(GF(5^2))
sage: B = GF(5^12).over(A)
sage: A.is_defined_over(GF(5^2))
True
sage: A.is_defined_over(GF(5))
False
sage: # needs sage.rings.finite_rings
sage: B.is_defined_over(A)
True
sage: B.is_defined_over(GF(5^4))
True
sage: B.is_defined_over(GF(5^2))
True
sage: B.is_defined_over(GF(5))
False
```

Note that an extension is defined over itself:

```python
sage: A.is_defined_over(A)  # needs sage.rings.finite_rings
True
```

(continues on next page)
$$\text{True}$$

```python
sage: A.is_defined_over(GF(5^4)) # needs sage.rings.finite_rings
True
```

See also:

`base()`, `bases()`, `absolute_base()`

**is_field** (*proof=True*)

Return whether or not this extension is a field.

**INPUT:**

- *proof* – a boolean (default: `False`)

**EXAMPLES:**

```python
sage: K = GF(5^5).over() # over GF(5)
# needs sage.rings.finite_rings
sage: K.is_field() # needs sage.rings.finite_rings
True

sage: S.<x> = QQ[]
sage: A = S.over(QQ)
sage: A.is_field()
False

sage: B = A.fraction_field()
sage: B.is_field()
True
```

**is_finite_over** (*base=None*)

Return whether or not this extension is finite over *base* (as a module).

**INPUT:**

- *base* – a commutative ring (which might be itself an extension) or `None` (default: `None`)

**EXAMPLES:**

```python
sage: # needs sage.rings.finite_rings
sage: K = GF(5^2).over() # over GF(5)
sage: L = GF(5^4).over(K)
sage: L.is_finite_over(K)
True

sage: L.is_finite_over(GF(5))
True
```

If `base` is omitted, it is set to its default which is the base of the extension:

```python
sage: L.is_finite_over() # needs sage.rings.finite_rings
True
```
**is_free_over**(base=None)

Return True if this extension is free (as a module) over base

INPUT:

- base – a commutative ring (which might be itself an extension) or None (default: None)

EXAMPLES:

```python
sage: # needs sage.rings.finite_rings
sage: K = GF(5^2).over()  # over GF(5)
sage: L = GF(5^4).over(K)
sage: L.is_free_over(K)
True
sage: L.is_free_over(GF(5))
True
```

If base is omitted, it is set to its default which is the base of the extension:

```python
sage: L.is_free_over()  # needs sage.rings.finite_rings
True
```

**ngens**(base=None)

Return the number of generators of this extension over base.

INPUT:

- base – a commutative ring (which might be itself an extension) or None (default: None)

EXAMPLES:

```python
sage: # needs sage.rings.finite_rings
sage: K = GF(5^2).over()  # over GF(5)
sage: K.gens()
(z2,)
sage: K.ngens()
1
sage: L = GF(5^4).over(K)
sage: L.gens(GF(5))
(z4, z2)
sage: L.ngens(GF(5))
2
```

**print_options**(**options**)

Update the printing options of this extension.

INPUT:

- over – an integer or Infinity (default: 0); the maximum number of bases included in the printing of this extension
- base – a base over which this extension is finite free; elements in this extension will be printed as a linear combinaison of a basis of this extension over the given base

EXAMPLES:
Observe what happens when we modify the option `over`:

```python
sage: D
Field in d with defining polynomial
    x^2 + ((1 - a) + ((1 + 2*a) - b)*c + ((2 + a) + (1 - a)*b)*c^2)*x + c over its base
sage: D.print_options(over=2)
Field in d with defining polynomial x^2 + ((1 - a) + ((1 + 2*a) - b)*c + ((2 + a) + (1 - a)*b)*c^2)*x + c over
Field in c with defining polynomial x^3 + (1 + (2 - a)*b)*x^2 + (2 + 2*b)*x - b
Field in b with defining polynomial x^2 + (3 - a)*x + a over its base
sage: D.print_options(over=Infinity)
Field in d with defining polynomial x^2 + ((1 - a) + ((1 + 2*a) - b)*c + ((2 + a) + (1 - a)*b)*c^2)*x + c over
Field in c with defining polynomial x^3 + (1 + (2 - a)*b)*x^2 + (2 + 2*b)*x - b
Field in b with defining polynomial x^2 + (3 - a)*x + a over
Field in a with defining polynomial x^2 + 4*x + 2 over
Finite Field of size 5
```

Now the option `base`:

```python
sage: d^2
-c + ((-1 + a) + ((-1 + 3*a) + b)*c + ((3 - a) + (-1 + a)*b)*c^2)*d
sage: D.basis_over(B)
[1, c, c^2, d, c*d, c^2*d]
sage: D.print_options(base=B)
sage: d^2
-c + ((-1 + a)*d + ((-1 + 3*a) + b)*c*d + ((3 - a) + (-1 + a)*b)*c^2*d
sage: D.basis_over(A)
[1, b, c, b*c, c^2, b*c^2, d, b*d, c*d, b*c*d, c^2*d, b*c^2*d]
sage: D.print_options(base=A)
sage: d^2
-c + ((-1 + a)*d + ((-1 + 3*a)*c*d + b*c*d + (3 - a)*c^2*d + (-1 + a)*b*c^2*d
```

**random_element()**

Return a random element in this extension.

**EXAMPLES:**

```python
sage: K = GF(5^2).over()
   # over GF(5)
sage: x = K.random_element(); x  # random
```
3 + z^2
sage: x.parent()
Field in z^2 with defining polynomial x^2 + 4*x + 2 over its base
sage: x.parent() is K
True

relative_degree()
Return the degree of this extension over its base

EXAMPLES:

sage: A = GF(5^4).over(GF(5^2))
# needs sage.rings.finite_rings
sage: A.relative_degree()
# needs sage.rings.finite_rings
2

See also:
degree(), absolute_degree()
sage.rings.ring_extension.common_base(K, L, degree)
Return a common base on which K and L are defined.

INPUT:

- K – a commutative ring
- L – a commutative ring
- degree – a boolean; if true, return the degree of K and L over their common base

EXAMPLES:

sage: from sage.rings.ring_extension import common_base
sage: common_base(GF(5^3), GF(5^7), False)  # needs sage.rings.finite_rings
Finite Field of size 5
sage: common_base(GF(5^3), GF(5^7), True)  # needs sage.rings.finite_rings
(Finite Field of size 5, 3, 7)
sage: common_base(GF(5^3), GF(7^5), False)  # needs sage.rings.finite_rings
Traceback (most recent call last):
  ... Not Implemented: unable to find a common base

When degree is set to True, we only look up for bases on which both K and L are finite:

sage: S.<x> = QQ[]
sage: common_base(S, QQ, False)
Rational Field
sage: common_base(S, QQ, True)
Traceback (most recent call last):
... NotImplementedError: unable to find a common base

sage.rings.ring_extension.generators(ring, base)

Return the generators of ring over base.

INPUT:

• ring – a commutative ring
• base – a commutative ring

EXAMPLES:

```python
sage: from sage.rings.ring_extension import generators
sage: S.<x> = QQ[]
sage: T.<y> = S[

sage: generators(T, S)
(y,)
sage: generators(T, QQ)
(y, x)
```

sage.rings.ring_extension.tower_bases(ring, degree)

Return the list of bases of ring (including itself); if degree is True, restrict to finite extensions and return in addition the degree of ring over each base.

INPUT:

• ring – a commutative ring
• degree – a boolean

EXAMPLES:

```python
sage: from sage.rings.ring_extension import tower_bases
sage: S.<x> = QQ[]
sage: T.<y> = S[

sage: tower_bases(T, False)
([Univariate Polynomial Ring in y over
  Univariate Polynomial Ring in x over Rational Field,
  Univariate Polynomial Ring in x over Rational Field,
  Rational Field],
  [])
sage: tower_bases(T, True)
([Univariate Polynomial Ring in y over
  Univariate Polynomial Ring in x over Rational Field],
  [1])
```

```python
sage: K.<a> = Qq(5^2)  # needs sage.rings.padics
sage: L.<w> = K.extension(x^3 - 5)  # needs sage.rings.padics
sage: tower_bases(L, True)  # needs sage.rings.padics
([5-adic Eisenstein Extension Field in w defined by x^3 - 5 over its base field,
  ...]
```

(continues on next page)
sage.rings.ring_extension_variable_names(ring, base)

Return the variable names of the generators of ring over base.

INPUT:

• ring – a commutative ring
• base – a commutative ring

EXAMPLES:

```
sage: from sage.rings.ring_extension import variable_names
sage: S.<x> = QQ[]
sage: T.<y> = S[]

sage: variable_names(T, S)
('y',)
sage: variable_names(T, QQ)
('y', 'x')
```

### 7.2 Elements lying in extension of rings

AUTHOR:

• Xavier Caruso (2019)

class sage.rings.ring_extension_element.RingExtensionElement

Bases: CommutativeAlgebraElement

Generic class for elements lying in ring extensions.

additive_order()

Return the additive order of this element.

EXAMPLES:

```
sage: K.<a> = GF(5^4).over(GF(5^2))
# needs sage.rings.finite_rings
sage: a.additive_order()  # needs sage.rings.finite_rings
5
```

backend(force=False)

Return the backend of this element.

INPUT:

• force – a boolean (default: False); if False, raise an error if the backend is not exposed

EXAMPLES:
```python
sage: # needs sage.rings.finite_rings
sage: F = GF(5^2)

sage: K.<z> = GF(5^4).over(F)

sage: x = z^10

sage: x
(3*z^2 + 2) + (3*z^2 + 1)*z

sage: y = x.backend()

sage: y
4*z^4 + 2*z^2 + 4*z + 4

sage: y.parent()
Finite Field in z4 of size 5^4
```

**in_base()**

Return this element as an element of the base.

**EXAMPLES:**

```python
sage: # needs sage.rings.finite_rings
sage: F = GF(5^2)

sage: K.<z> = GF(5^4).over(F)

sage: x = z^3 + z^2 + z + 4

sage: y = x.in_base()

sage: y
z^2 + 1

sage: y.parent()
Finite Field in z2 of size 5^2
```

When the element is not in the base, an error is raised:

```python
sage: z.in_base() # needs sage.rings.finite_rings

Traceback (most recent call last):
...
ValueError: z is not in the base
```

```python
sage: # needs sage.rings.finite_rings
sage: S.<X> = F[]

sage: E = S.over(F)

sage: f = E(1)

sage: g = f.in_base(); g
1

sage: g.parent()
Finite Field in z2 of size 5^2
```

**is_nilpotent()**

Return whether if this element is nilpotent in this ring.

**EXAMPLES:**

```python
sage: A.<x> = PolynomialRing(QQ)

sage: E = A.over(QQ)

sage: E(0).is_nilpotent()
True
```

(continues on next page)
### is_prime()

Return whether this element is a prime element in this ring.

**EXAMPLES:**

```python
sage: A.<x> = PolynomialRing(QQ)
sage: E = A.over(QQ)
sage: E(x^2 + 1).is_prime()  # needs sage.libs.pari
True
sage: E(x^2 - 1).is_prime()  # needs sage.libs.pari
False
```

### is_square(root=False)

Return whether this element is a square in this ring.

**INPUT:**

- **root** – a boolean (default: False); if True, return also a square root

**EXAMPLES:**

```python
sage: # needs sage.rings.finite_rings
sage: K.<a> = GF(5^3).over()
sage: a.is_square()  # False
sage: a.is_square(root=True)  # (False, None)
sage: b = a + 1
sage: b.is_square()  # True
sage: b.is_square(root=True)  # (True, 2 + 3*a + a^2)
```

### is_unit()

Return whether if this element is a unit in this ring.

**EXAMPLES:**

```python
sage: A.<x> = PolynomialRing(QQ)
sage: E = A.over(QQ)
sage: E(4).is_unit()  # True
sage: E(x).is_unit()  # False
```

### multiplicative_order()

Return the multiplicative order of this element.

**EXAMPLES:**
sage: K.<a> = GF(5^4).over(GF(5^2))  # needs sage.rings.finite_rings
sage: a.multiplicative_order()  # needs sage.rings.finite_rings
624

\texttt{sqrt}(\texttt{extend=True, all=False, name=None})

Return a square root or all square roots of this element.

\textbf{INPUT}:

- \texttt{extend} – a boolean (default: \texttt{True}); if “\texttt{True}”, return a square root in an extension ring, if necessary. Otherwise, raise a \texttt{ValueError} if the root is not in the ring
- \texttt{all} – a boolean (default: \texttt{False}); if \texttt{True}, return all square roots of this element, instead of just one.
- \texttt{name} – Required when \texttt{extend=True} and \texttt{self} is not a square. This will be the name of the generator extension.

\textbf{Note}: The option \texttt{extend=True} is often not implemented.

\textbf{EXAMPLES}:

\begin{verbatim}
sage: # needs sage.rings.finite_rings
e: K.<a> = GF(5^3).over()
sage: b = a + 1
sage: b.sqrt()
2 + 3*a + a^2
sage: b.sqrt(all=True)
[2 + 3*a + a^2, 3 + 2*a - a^2]
\end{verbatim}

class sage.rings.ring_extension_element.
\texttt{RingExtensionFractionFieldElement}

Bases: \texttt{RingExtensionElement}

A class for elements lying in fraction fields of ring extensions.

def\texttt{denominator}()

Return the denominator of this element.

\textbf{EXAMPLES}:

\begin{verbatim}
sage: # needs sage.rings.number_field
sage: R.<x> = ZZ[]
sage: A.<a> = ZZ.extension(x^2 - 2)
sage: OK = A.over()  # over ZZ
sage: K = OK.fraction_field(); K
Fraction Field of Order in Number Field in a with defining polynomial x^2 - 2 over its base
sage: x = K(1/a); x
a/2
sage: denom = x.denominator(); denom
2
\end{verbatim}

The denominator is an element of the ring which was used to construct the fraction field:
sage: denom.parent()  # needs sage.rings.number_field
Order in Number Field in a with defining polynomial x^2 - 2 over its base
sage: denom.parent() is OK  # needs sage.rings.number_field
True

\textbf{numerator()}

Return the numerator of this element.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: # needs sage.rings.number_field
sage: x = polygen(ZZ,'x')
sage: A.<a> = ZZ.extension(x^2 - 2)
sage: OK = A.over()  # over ZZ
sage: K = OK.fraction_field(); K
Fraction Field of Order in Number Field in a with defining polynomial x^2 - 2 over its base
sage: x = K(1/a); x
a/2
sage: num = x.numerator(); num
a
\end{verbatim}

The numerator is an element of the ring which was used to construct the fraction field:

\begin{verbatim}
sage: num.parent()  # needs sage.rings.number_field
Order in Number Field in a with defining polynomial x^2 - 2 over its base
sage: num.parent() is OK  # needs sage.rings.number_field
True
\end{verbatim}

\textbf{class sage.rings.ring_extension_element.RingExtensionWithBasisElement}

Bases: \textit{RingExtensionElement}

A class for elements lying in finite free extensions.

\textbf{charpoly}(\textit{base=None, var='x'})

Return the characteristic polynomial of this element over \textit{base}.

\textbf{INPUT:}

\begin{itemize}
  \item \textit{base} – a commutative ring (which might be itself an extension) or \texttt{None}
\end{itemize}

\textbf{EXAMPLES:}

\begin{verbatim}
sage: # needs sage.rings.finite_rings
sage: F = GF(5)
sage: K.<a> = GF(5^3).over(F)
sage: L.<b> = GF(5^6).over(K)
sage: u = a/(1+b)
sage: chi = u.charpoly(K); chi
x^2 + (1 + 2*a + 3*a^2)*x + 3 + 2*a^2
\end{verbatim}

We check that the charpoly has coefficients in the base ring:
and that it annihilates u:

```
sage: chi(u)  # needs sage.rings.finite_rings
0
```

Similarly, one can compute the characteristic polynomial over F:

```
sage: u.charpoly(F)  # needs sage.rings.finite_rings
x^6 + x^4 + 2*x^3 + 3*x + 4
```

A different variable name can be specified:

```
sage: u.charpoly(\text{var}='t')  # needs sage.rings.finite_rings
t^6 + t^4 + 2*t^3 + 3*t + 4
```

If base is omitted, it is set to its default which is the base of the extension:

```
sage: u.charpoly()  # needs sage.rings.finite_rings
x^2 + (1 + 2*a + 3*a^2)*x + 3 + 2*a^2
```

Note that \texttt{base} must be an explicit base over which the extension has been defined (as listed by the method \texttt{bases()}):

```
sage: u.charpoly(GF(5^2))  # needs sage.rings.finite_rings
Traceback (most recent call last):
...
ValueError: not (explicitly) defined over Finite Field in z2 of size 5^2
```

\textbf{matrix(\texttt{base=None})}

Return the matrix of the multiplication by this element (in the basis output by \texttt{basis_over()}).

\textbf{INPUT}:

\begin{itemize}
  \item \texttt{base} – a commutative ring (which might be itself an extension) or \texttt{None}
\end{itemize}

\textbf{EXAMPLES}:

```
sage: # needs sage.rings.finite_rings
sage: K.<a> = GF(5^3).over()  # over GF(5)
sage: L.<b> = GF(5^6).over(K)
sage: u = a/(1+b)
sage: u
(2 + a + 3*a^2) + (3 + 3*a + a^2)*b
```

(continues on next page)
(continued from previous page)

```
sage: b*u
(3 + 2*a^2) + (2 + 2*a - a^2)*b
sage: u.matrix(K)
[2 + a + 3*a^2 3 + 3*a + a^2]
[3 + 2*a^2 2 + 2*a - a^2]
sage: u.matrix(GF(5))
[2 1 3 3 3 1]
[1 3 1 2 0 3]
[2 3 3 1 3 0]
[3 0 2 2 2 4]
[4 2 0 3 0 2]
[0 4 2 4 2 0]
```

If `base` is omitted, it is set to its default which is the base of the extension:

```
sage: u.matrix()  # needs sage.rings.finite_rings
[2 + a + 3*a^2 3 + 3*a + a^2]
[3 + 2*a^2 2 + 2*a - a^2]
```

Note that `base` must be an explicit base over which the extension has been defined (as listed by the method `bases()`):

```
sage: u.matrix(GF(5^2))  # needs sage.rings.finite_rings
Traceback (most recent call last):
...  
ValueError: not (explicitly) defined over Finite Field in z2 of size 5^2
```

`minpoly(base=None, var='x')`

Return the minimal polynomial of this element over `base`.

**INPUT:**

- `base` – a commutative ring (which might be itself an extension) or `None`

**EXAMPLES:**

```
sage: # needs sage.rings.finite_rings
sage: F = GF(5)
sage: K.<a> = GF(5^3).over(F)
sage: L.<b> = GF(5^6).over(K)
sage: u = 1 / (a+b)
sage: chi = u.minpoly(); chi
x^2 + (2*a + a^2)*x - 1 + a
```

We check that the minimal polynomial has coefficients in the base ring:

```
sage: chi.base_ring()  # needs sage.rings.finite_rings
Field in a with defining polynomial x^3 + 3*x + 3 over its base
sage: chi.base_ring() is K  # needs sage.rings.finite_rings
True
```
and that it annihilates $u$:

```
sage: chi(u)  # needs sage.rings.finite_rings
0
```

Similarly, one can compute the minimal polynomial over $F$:

```
sage: u.minpoly(F)  # needs sage.rings.finite_rings
x^6 + 4*x^5 + x^4 + 2*x^2 + 3
```

A different variable name can be specified:

```
sage: u.minpoly(F, var='t')  # needs sage.rings.finite_rings
t^6 + 4*t^5 + t^4 + 2*t^2 + 3
```

If `base` is omitted, it is set to its default which is the base of the extension:

```
sage: u.minpoly()  # needs sage.rings.finite_rings
x^2 + (2*a + a^2)*x - 1 + a
```

Note that `base` must be an explicit base over which the extension has been defined (as listed by the method `bases()`):

```
sage: u.minpoly(GF(5^2))  # needs sage.rings.finite_rings
Traceback (most recent call last):
... ValueError: not (explicitly) defined over Finite Field in z2 of size 5^2
```

**norm**

Return the norm of this element over base.

**INPUT:**

- `base` – a commutative ring (which might be itself an extension) or `None`

**EXAMPLES:**

```
sage: # needs sage.rings.finite_rings
sage: F = GF(5)
sage: K.<a> = GF(5^3).over(F)
sage: L.<b> = GF(5^6).over(K)
sage: u = a/(1+b)
sage: nr = u.norm(K); nr
3 + 2*a^2
```

We check that the norm lives in the base ring:

```
sage: nr.parent()  # needs sage.rings.finite_rings
Field in a with defining polynomial x^3 + 3*x + 3 over its base
sage: nr.parent() is K  # needs sage.rings.finite_rings
```

(continues on next page)
Similarly, one can compute the norm over F:

```
sage: u.norm(F)
needs sage.rings.finite_rings
4
```

We check the transitivity of the norm:

```
sage: u.norm(F) == nr.norm(F)
needs sage.rings.finite_rings
True
```

If `base` is omitted, it is set to its default which is the base of the extension:

```
sage: u.norm()
needs sage.rings.finite_rings
3 + 2*a^2
```

Note that `base` must be an explicit base over which the extension has been defined (as listed by the method `bases()`):

```
sage: u.norm(GF(5^2))
needs sage.rings.finite_rings
Traceback (most recent call last):
... ValueError: not (explicitly) defined over Finite Field in z2 of size 5^2
```

**polynomial** (`base=base, var='x'`)

Return a polynomial (in one or more variables) over `base` whose evaluation at the generators of the parent equals this element.

**INPUT:**

- `base` – a commutative ring (which might be itself an extension) or `None`

**EXAMPLES:**

```
sage: F.<a> = GF(5^2).over()  # over GF(5)
sage: K.<b> = GF(5^4).over(F)
sage: L.<c> = GF(5^12).over(K)
sage: u = 1/(a + b + c); u
(2 + (-1 - a)*b) + ((2 + 3*a) + (1 - a)*b)*c + ((-1 - a) - a*b)*c^2
sage: P = u.polynomial(K); P
((-1 - a) - a*b)*x^2 + ((2 + 3*a) + (1 - a)*b)*x + 2 + (-1 - a)*b
sage: P.base_ring() is K
True
sage: P(c) == u
True
```

When the base is `F`, we obtain a bivariate polynomial:
General Rings, Ideals, and Morphisms, Release 10.2

sage: P = u.polynomial(F); P
˓→ needs sage.rings.finite_rings
(-a)*x0^2*x1 + (-1 - a)*x0^2 + (1 - a)*x0*x1 + (2 + 3*a)*x0 + (-1 - a)*x1 + 2

We check that its value at the generators is the element we started with:

sage: L.gens(F)
˓→ needs sage.rings.finite_rings
(c, b)
sage: P(c, b) == u
˓→ needs sage.rings.finite_rings
True

Similarly, when the base is GF(5), we get a trivariate polynomial:

sage: P = u.polynomial(GF(5)); P # needs sage.rings.finite_rings
-x0^2*x1*x2 - x0^2*x2 - x0*x1*x2 - x0^2 + x0*x1 - 2*x0*x2 - x1*x2 + 2*x0 - x1 + 2

sage: P(c, b, a) == u # needs sage.rings.finite_rings
True

Different variable names can be specified:

sage: u.polynomial(GF(5), var='y') # needs sage.rings.finite_rings
-y0^2*y1*y2 - y0^2*y2 - y0*y1*y2 - y0^2 + y0*y1 - 2*y0*y2 - y1*y2 + 2*y0 - y1 + 2

sage: u.polynomial(GF(5), var=['x', 'y', 'z']) # needs sage.rings.finite_rings
-x^2*y^z - x^2*z - x*y*z - x^2 + x*y - 2*x*z - y^z + 2*x - y + 2

If base is omitted, it is set to its default which is the base of the extension:

sage: u.polynomial() # needs sage.rings.finite_rings
((-1 - a) - a*b)*x^2 + ((2 + 3*a) + (1 - a)*b)*x + 2 + (-1 - a)*b

Note that base must be an explicit base over which the extension has been defined (as listed by the method bases()):

sage: u.polynomial(GF(5^3)) # needs sage.rings.finite_rings
Traceback (most recent call last):
... ValueError: not (explicitly) defined over Finite Field in z3 of size 5^3

trace(base=None)

Return the trace of this element over base.

INPUT:

• base – a commutative ring (which might be itself an extension) or None

EXAMPLES:

sage: # needs sage.rings.finite_rings
sage: F = GF(5)
sage: K.<a> = GF(5^3).over(F)
We check that the trace lives in the base ring:

```
sage: tr.parent()  # needs sage.rings.finite_rings
Field in a with defining polynomial x^3 + 3*x + 3 over its base
sage: tr.parent() is K  # needs sage.rings.finite_rings
True
```

Similarly, one can compute the trace over F:

```
sage: u.trace(F)  # needs sage.rings.finite_rings
0
```

We check the transitivity of the trace:

```
sage: u.trace(F) == tr.trace(F)  # needs sage.rings.finite_rings
True
```

If base is omitted, it is set to its default which is the base of the extension:

```
sage: u.trace()  # needs sage.rings.finite_rings
-1 + 3*a + 2*a^2
```

Note that base must be an explicit base over which the extension has been defined (as listed by the method bases()):

```
sage: u.trace(GF(5^2))  # needs sage.rings.finite_rings
Traceback (most recent call last):
  ...
ValueError: not (explicitly) defined over Finite Field in z2 of size 5^2
```

**vector(base=None)**

Return the vector of coordinates of this element over base (in the basis output by the method basis_over()).

**INPUT:**

- base – a commutative ring (which might be itself an extension) or None

**EXAMPLES:**

```
sage: # needs sage.rings.finite_rings
sage: F = GF(5)
sage: K.<a> = GF(5^2).over()  # over F
```

(continues on next page)
sage: L.<b> = GF(5^6).over(K)
sage: x = (a+b)^4; x
(-1 + a) + (3 + a)*b + (1 - a)*b^2
sage: x.vector(K)  # basis is (1, b, b^2)
(-1 + a, 3 + a, 1 - a)
sage: x.vector(F)  # basis is (1, a, b, a*b, b^2, a*b^2)
(4, 1, 3, 1, 1, 4)

If base is omitted, it is set to its default which is the base of the extension:

sage: x.vector()  # needs sage.rings.finite_rings
(-1 + a, 3 + a, 1 - a)

Note that base must be an explicit base over which the extension has been defined (as listed by the method bases()):

sage: x.vector(GF(5^3))  # needs sage.rings.finite_rings
Traceback (most recent call last):
... ValueError: not (explicitly) defined over Finite Field in z3 of size 5^3

7.3 Morphisms between extension of rings

AUTHOR:
- Xavier Caruso (2019)

class sage.rings.ring_extension_morphism.MapFreeModuleToRelativeRing
  Bases: Map
  Base class of the module isomorphism between a ring extension and a free module over one of its bases.

  is_injective()
  Return whether this morphism is injective.
  
  EXAMPLES:

  sage: K = GF(11^6).over(GF(11^3))  # needs sage.rings.finite_rings
  sage: V, i, j = K.free_module()  # needs sage.rings.finite_rings
  sage: i.is_injective()  # needs sage.rings.finite_rings
  True

  is_surjective()
  Return whether this morphism is surjective.
  
  EXAMPLES:
class sage.rings.ring_extension_morphism.MapRelativeRingToFreeModule

Bases: Map

Base class of the module isomorphism between a ring extension and a free module over one of its bases.

is_injective()

Return whether this morphism is injective.

EXAMPLES:

```sage
sage: K = GF(11^6).over(GF(11^3))
# needs sage.rings.finite_rings
sage: V, i, j = K.free_module()
# needs sage.rings.finite_rings
sage: j.is_injective()
# needs sage.rings.finite_rings
True
```

is_surjective()

Return whether this morphism is injective.

EXAMPLES:

```sage
sage: K = GF(11^6).over(GF(11^3))
# needs sage.rings.finite_rings
sage: V, i, j = K.free_module()
# needs sage.rings.finite_rings
sage: j.is_surjective()
# needs sage.rings.finite_rings
True
```

class sage.rings.ring_extension_morphism.RingExtensionBackendIsomorphism

Bases: RingExtensionHomomorphism

A class for implementing isomorphisms taking an element of the backend to its ring extension.

class sage.rings.ring_extension_morphism.RingExtensionBackendReverseIsomorphism

Bases: RingExtensionHomomorphism

A class for implementing isomorphisms from a ring extension to its backend.

class sage.rings.ring_extension_morphism.RingExtensionHomomorphism

Bases: RingMap

A class for ring homomorphisms between extensions.

base_map()

Return the base map of this morphism or just None if the base map is a coercion map.

EXAMPLES:
We define the absolute Frobenius of L:

```python
tsage: FrobL = L.hom([b^5, a^5]); FrobL
Ring endomorphism of Field in b with defining polynomial x^3 + (2 + 2*a)*x - a over its base
  Defn: b |--> (-1 + a) + (1 + 2*a)*b + a*b^2
  with map on base ring:
  a |--> 1 - a
tsage: FrobL.base_map()
Ring morphism:
  From: Field in a with defining polynomial x^2 + 4*x + 2 over its base
  To: Field in b with defining polynomial x^3 + (2 + 2*a)*x - a over its base
  Defn: a |--> 1 - a
```

The square of FrobL acts trivially on K; in other words, it has a trivial base map:

```python
tsage: phi = FrobL^2; phi
Ring endomorphism of Field in b with defining polynomial x^3 + (2 + 2*a)*x - a over its base
  Defn: b |--> 2 + 2*a*b + (2 - a)*b^2
tsage: phi.base_map()
```

is_identity() returning whether this morphism is the identity.

EXAMPLES:

```python
tsage: # needs sage.rings.finite_rings
sage: K.<a> = GF(5^2).over(F)  # over GF(5)
sage: FrobK = K.hom([a^5])
sage: FrobK.is_identity()
False
tsage: (FrobK^2).is_identity()
True
```

Coercion maps are not considered as identity morphisms:
is_injective()  
Return whether this morphism is injective.

EXAMPLES:

    sage: # needs sage.rings.finite_rings
    sage: K = GF(5^10).over(GF(5^5))
    sage: iota = K.defining_morphism(); iota
    Ring morphism:
    From: Finite Field in z5 of size 5^5
    To: Field in z10 with defining polynomial
       x^2 + (2*z5^3 + 2*z5^2 + 4*z5 + 4)*x + z5 over its base
    Defn: z5 |--> z5
    sage: iota.is_injective()
    True
    sage: K = GF(7).over(ZZ)
    sage: iota = K.defining_morphism(); iota
    Ring morphism:
    From: Integer Ring
    To: Finite Field of size 7 over its base
    Defn: 1 |--> 1
    sage: iota.is_injective()
    False

is_surjective()  
Return whether this morphism is surjective.

EXAMPLES:

    sage: # needs sage.rings.finite_rings
    sage: K = GF(5^10).over(GF(5^5))
    sage: iota = K.defining_morphism(); iota
    Ring morphism:
    From: Finite Field in z5 of size 5^5
    To: Field in z10 with defining polynomial
       x^2 + (2*z5^3 + 2*z5^2 + 4*z5 + 4)*x + z5 over its base
    Defn: z5 |--> z5
    sage: iota.is_surjective()
    False
    sage: K = GF(7).over(ZZ)
    sage: iota = K.defining_morphism(); iota
    Ring morphism:
    From: Integer Ring
    To: Finite Field of size 7 over its base
    Defn: 1 |--> 1
    sage: iota.is_surjective()
    True
8.1 Generic data structures and algorithms for rings

AUTHORS:

- Lorenz Panny (2022): ProductTree, prod_with_derivative()

**class** sage.rings.generic.ProductTree(leave)

**Bases:** object

A simple binary product tree, i.e., a tree of ring elements in which every node equals the product of its children.
(In particular, the root equals the product of all leaves.)

Product trees are a very useful building block for fast computer algebra. For example, a quasilinear-time Discrete Fourier Transform (the famous Fast Fourier Transform) can be implemented as follows using the remainders() method of this class:

```sage
from sage.rings.generic import ProductTree
F = GF(65537)
a = F(1111)
assert a.multiplicative_order() == 1024
R.<x> = F[]
ms = [x - a^i for i in range(1024)]  # roots of unity
ys = [F.random_element() for _ in range(1024)]  # input vector
zs = ProductTree(ms).remainders(R(ys))  # compute FFT!
zs == [R(ys) % m for m in ms]
True
```

This class encodes the tree as layers: Layer 0 is just a tuple of the leaves. Layer \( i + 1 \) is obtained from layer \( i \) by replacing each pair of two adjacent elements by their product, starting from the left. (If the length is odd, the unpaired element at the end is simply copied as is.) This iteration stops as soon as it yields a layer containing only a single element (the root).

**Note:** Use this class if you need the remainders() method. To compute just the product, prod() is likely faster.

**INPUT:**

- leaves – an iterable of elements in a common ring

**EXAMPLES:**
We can access the individual layers of the tree:

```
sage: tree.layers
 (6, 35, 143, 323, 667, 1147, 1763, 2491, 3599, 4757, 5767, 7387, 97),
 (210, 46189, 765049, 4391633, 17120443, 42600829, 97),
 (9699690, 3359814435017, 729345064647247, 97),
 (32589158477190044730, 70746471270782959),
 (2305567963945518424753102147331756070,)]
```

```
leaves()
```

Return a tuple containing the leaves of this product tree.

EXAMPLES:

```
sage: from sage.rings.generic import ProductTree
sage: R.<x> = GF(101)[]
sage: vs = [x - i for i in range(1,10)]
sage: tree = ProductTree(vs)
sage: tree.leaves()
(x + 100, x + 99, x + 98, ..., x + 93, x + 92)
sage: tree.leaves() == tuple(vs)
True
```

```
remainders(x)
```

Given a value \( x \), return a list of all remainders of \( x \) modulo the leaves of this product tree.

The base ring must support the \% operator for this method to work.

INPUT:

- \( x \) – an element of the base ring of this product tree

EXAMPLES:

```
sage: from sage.rings.generic import ProductTree
sage: R.<x> = GF(101)[]
sage: vs = [x - i for i in range(1,10)]
sage: tree = ProductTree(vs)
sage: tree.remainders(x^7 + x + 1)
[3, 30, 70, 27, 58, 72, 98, 98, 23]
sage: tree.remainders(x^100)
[1, 1, 1, 1, 1, 1, 1, 1, 1]
sage: vs = prime_range(100)
sage: tree = ProductTree(vs)
sage: tree.remainders(3599)
[1, 2, 4, 1, 2, 11, 12, 8, 11, 3, 3, 10, 32, 30, 27, 48, 0, 0, 48, 49, 22, 44, 30, 39, 10]
```

```
```
sage: from sage.rings.generic import ProductTree
sage: vs = prime_range(100)
sage: tree = ProductTree(vs)
sage: n = 1085749272377676749812331719267
sage: tree.remainders(n)
[1, 1, 2, 1, 9, 1, 7, 15, 8, 20, 15, 6, 27, 11, 2, 6, 0, 25, 49, 5, 51, 4, 19, → 74, 13]
sage: [n % v for v in vs]
[1, 1, 2, 1, 9, 1, 7, 15, 8, 20, 15, 6, 27, 11, 2, 6, 0, 25, 49, 5, 51, 4, 19, → 74, 13]

root()

Return the value at the root of this product tree (i.e., the product of all leaves).

EXAMPLES:

sage: from sage.rings.generic import ProductTree
sage: R.<x> = GF(101)[]
sage: vs = [x - i for i in range(1,10)]
sage: tree = ProductTree(vs)
sage: tree.root()
x^9 + 56*x^8 + 62*x^7 + 44*x^6 + 47*x^5 + 42*x^4 + 15*x^3 + 11*x^2 + 12*x + 13
sage: tree.root() == prod(vs)
True

sage.rings.generic.prod_with_derivative(pairs)

Given an iterable of pairs \((f, \partial f)\) of ring elements, return the pair \((\prod f, \partial \prod f)\), assuming \(\partial\) is an operator obeying the standard product rule.

This function is entirely algebraic, hence still works when the elements \(f\) and \(\partial f\) are all passed through some ring homomorphism first. One particularly useful instance of this is evaluating the derivative of a product of polynomials at a point without fully expanding the product; see the second example below.

INPUT:

• pairs – an iterable of tuples \((f, \partial f)\) of elements of a common ring

ALGORITHM: Repeated application of the product rule.

EXAMPLES:

sage: from sage.rings.generic import prod_with_derivative
sage: R.<x> = ZZ[

The main reason for this function to exist is that it allows us to evaluate the derivative of a product of polynomials at a point \(\alpha\) without ever fully expanding the product as a polynomial:
sage: alpha = 42
sage: F(alpha)
442943981574522759
sage: dF(alpha)
104645261461514994
sage: us = [f(alpha) for f in fs]
sage: vs = [f.derivative()(alpha) for f in fs]
sage: prod_with_derivative(zip(us, vs))
(442943981574522759, 104645261461514994)
9.1 Big O for various types (power series, p-adics, etc.)

See also:
- asymptotic expansions
- p-adic numbers
- power series
- polynomials

\texttt{sage.rings.big_oh.O(*x, **kwds)}

Big O constructor for various types.

**EXAMPLES:**

This is useful for writing power series elements:

\begin{verbatim}
sage: R.<t> = ZZ[['t']]
sage: (1+t)^10 + O(t^5)
1 + 10*t + 45*t^2 + 120*t^3 + 210*t^4 + O(t^5)
\end{verbatim}

A power series ring is created implicitly if a polynomial element is passed:

\begin{verbatim}
sage: R.<x> = QQ['x']
sage: O(x^100)
O(x^100)
sage: 1/(1+x+O(x^5))
1 - x + x^2 - x^3 + x^4 + O(x^5)
sage: R.<u,v> = QQ[]
sage: 1 + u + v^2 + O(u, v)^5
1 + u + v^2 + O(u, v)^5
\end{verbatim}

This is also useful to create $p$-adic numbers:

\begin{verbatim}
sage: O(7^6)  # needs sage.rings.padics
0(7^6)
sage: 1/3 + O(7^6)  # needs sage.rings.padics
5 + 4*7 + 4*7^2 + 4*7^3 + 4*7^4 + 4*7^5 + O(7^6)
\end{verbatim}

It behaves well with respect to adding negative powers of $p$: 
There are problems if you add a rational with very negative valuation to an $O$-Term:

\begin{verbatim}
 sage: 11^-12 + O(11^15)  # needs sage.rings.padics
 11^-12 + 0(11^8)
\end{verbatim}

The reason that this fails is that the constructor doesn’t know the right precision cap to use. If you cast explicitly or use other means of element creation, you can get around this issue:

\begin{verbatim}
 sage: # needs sage.rings.padics
 sage: K = Qp(11, 30)
sage: K(11^-12) + O(11^15)  
11^-12 + O(11^15)
sage: 11^-12 + K(O(11^15))  
11^-12 + 0(11^15)
sage: K(11^-12, absprec=15)  
11^-12 + 0(11^15)
sage: K(11^-12, 15)  
11^-12 + 0(11^15)
\end{verbatim}

We can also work with asymptotic expansions:

\begin{verbatim}
 sage: A.<n> = AsymptoticRing(growth_group='QQ^n * n^QQ * log(n)^QQ',  # needs sage.symbolic
 ....: coefficient_ring=QQ); A
Asymptotic Ring <QQ^n * n^QQ * log(n)^QQ * Signs^n> over Rational Field
 sage: 0(n)  # needs sage.symbolic
 0(n)
\end{verbatim}

Application with Puiseux series:

\begin{verbatim}
 sage: P.<y> = PuiseuxSeriesRing(ZZ)
sage: y^(1/5) + O(y^(1/3))
y^(1/5) + O(y^(1/3))
 sage: y^(1/3) + O(y^(1/5))
 0(y^(1/5))
\end{verbatim}
9.2 Signed and Unsigned Infinities

The unsigned infinity “ring” is the set of two elements

1. infinity
2. A number less than infinity

The rules for arithmetic are that the unsigned infinity ring does not canonically coerce to any other ring, and all other rings canonically coerce to the unsigned infinity ring, sending all elements to the single element “a number less than infinity” of the unsigned infinity ring. Arithmetic and comparisons then take place in the unsigned infinity ring, where all arithmetic operations that are well-defined are defined.

The infinity “ring” is the set of five elements

1. plus infinity
2. a positive finite element
3. zero
4. a negative finite element
5. negative infinity

The infinity ring coerces to the unsigned infinity ring, sending the infinite elements to infinity and the non-infinite elements to “a number less than infinity.” Any ordered ring coerces to the infinity ring in the obvious way.

**Note:** The shorthand oo is predefined in Sage to be the same as +Infinity in the infinity ring. It is considered equal to, but not the same as Infinity in the UnsignedInfinityRing.

**EXAMPLES:**

We fetch the unsigned infinity ring and create some elements:

```sage
sage: P = UnsignedInfinityRing; P
The Unsigned Infinity Ring
sage: P(5)
A number less than infinity
sage: P.ngens()
1
sage: unsigned_oo = P.0; unsigned_oo
Infinity
```

We compare finite numbers with infinity:

```sage
sage: 5 < unsigned_oo
True
sage: 5 > unsigned_oo
False
sage: unsigned_oo < 5
False
sage: unsigned_oo > 5
True
```

Demonstrating the shorthand oo versus Infinity:
We do arithmetic:

\[
\text{sage: unsigned_oo + 5}
\]

\[
\text{Infinity}
\]

We make \(1 / \text{unsigned_oo}\) return the integer 0 so that arithmetic of the following type works:

\[
\text{sage: (1/unsigned_oo) + 2}
\]

\[
2
\]

\[
\text{sage: 32/5 - (2.439/unsigned_oo)}
\]

\[
32/5
\]

Note that many operations are not defined, since the result is not well-defined:

\[
\text{sage: unsigned_oo/0}
\]

```
Traceback (most recent call last):
...
ValueError: quotient of number < oo by number < oo not defined
```

What happened above is that 0 is canonically coerced to “A number less than infinity” in the unsigned infinity ring. Next, Sage tries to divide by multiplying with its inverse. Finally, this inverse is not well-defined.

\[
\text{sage: 0/unsigned_oo}
\]

\[
0
\]

\[
\text{sage: unsigned_oo * 0}
\]

```
Traceback (most recent call last):
...
ValueError: unsigned oo times smaller number not defined
```

In the infinity ring, we can negate infinity, multiply positive numbers by infinity, etc.

\[
\text{sage: P = InfinityRing; P}
\]

```
The Infinity Ring
```

\[
\text{sage: P(5)}
\]

```
A positive finite number
```

The symbol oo is predefined as a shorthand for +Infinity:

\[
\text{sage: oo}
\]

\[
+Infinity
\]

We compare finite and infinite elements:
We can do more arithmetic than in the unsigned infinity ring:

```
sage: 2 * oo
+Infinity
sage: -2 * oo
-Infinity
sage: 1 - oo
-Infinity
sage: 1 / oo
0
sage: -1 / oo
0
```

We make $1 / oo$ and $1 / -oo$ return the integer 0 instead of the infinity ring Zero so that arithmetic of the following type works:

```
sage: (1/oo) + 2
2
sage: 32/5 - (2.439/-oo)
32/5
```

If we try to subtract infinities or multiply infinity by zero we still get an error:

```
sage: oo - oo
Traceback (most recent call last):
  ...  
SignError: cannot add infinity to minus infinity
sage: 0 * oo
Traceback (most recent call last):
  ...  
SignError: cannot multiply infinity by zero
sage: P(2) + P(-3)
Traceback (most recent call last):
  ...  
SignError: cannot add positive finite value to negative finite value
```

Signed infinity can also be represented by RR / RDF elements. But unsigned infinity cannot:

```
sage: oo in RR, oo in RDF
(True, True)
sage: unsigned_infinity in RR, unsigned_infinity in RDF
(False, False)
```

**class** `sage.rings.infinity.AnInfinity`

Bases: `object`

---

9.2. Signed and Unsigned Infinities
\texttt{lcm}(x)

Return the least common multiple of oo and x, which is by definition oo unless x is 0.

EXAMPLES:

\begin{verbatim}
sage: oo.lcm(0)
0
sage: oo.lcm(oo)
+Infinity
sage: oo.lcm(-oo)
+Infinity
sage: oo.lcm(10)
+Infinity
sage: (-oo).lcm(10)
+Infinity
\end{verbatim}

\textbf{class} sage.rings.infinity.\texttt{FiniteNumber}(parent, x)

\textbf{Bases:} RingElement

Initialize self.

\texttt{sign}()

Return the sign of self.

EXAMPLES:

\begin{verbatim}
sage: sign(InfinityRing(2))
1
sage: sign(InfinityRing(0))
0
sage: sign(InfinityRing(-2))
-1
\end{verbatim}

\texttt{sqrt}()

EXAMPLES:

\begin{verbatim}
sage: InfinityRing(7).sqrt()
A positive finite number
sage: InfinityRing(0).sqrt()
Zero
sage: InfinityRing(-0.001).sqrt()
Traceback (most recent call last):
...
SignError: cannot take square root of a negative number
\end{verbatim}

\textbf{class} sage.rings.infinity.\texttt{InfinityRing_class}

\textbf{Bases:} Singleton, Ring

Initialize self.

\texttt{fraction_field}()

This isn’t really a ring, let alone an integral domain.

\texttt{gen}(n=0)

The two generators are plus and minus infinity.

EXAMPLES:
sage: InfinityRing.gen(0)
+Infinity
sage: InfinityRing.gen(1)
-Infinity
sage: InfinityRing.gen(2)
Traceback (most recent call last):
...
IndexError: n must be 0 or 1

gens()
The two generators are plus and minus infinity.

EXAMPLES:

sage: InfinityRing.gens()
[+Infinity, -Infinity]

is_commutative() The Infinity Ring is commutative

EXAMPLES:

sage: InfinityRing.is_commutative()
True

is_zero() The Infinity Ring is not zero

EXAMPLES:

sage: InfinityRing.is_zero()
False

ngens() The two generators are plus and minus infinity.

EXAMPLES:

sage: InfinityRing.ngens()
2
sage: len(InfinityRing.gens())
2

class sage.rings.infinity.LessThanInfinity(*args)
Bases: _uniq, RingElement
Initialize self.

EXAMPLES:

sage: sage.rings.infinity.LessThanInfinity() is UnsignedInfinityRing(5)
True

sign() Raise an error because the sign of self is not well defined.

EXAMPLES:
```python
sage: sign(UnsignedInfinityRing(2))
Traceback (most recent call last):
  ...  
NotImplementedError: sign of number < oo is not well defined
sage: sign(UnsignedInfinityRing(0))
Traceback (most recent call last):
  ...  
NotImplementedError: sign of number < oo is not well defined
sage: sign(UnsignedInfinityRing(-2))
Traceback (most recent call last):
  ...  
NotImplementedError: sign of number < oo is not well defined
```

```python
definition

class sage.rings.infinity.MinusInfinity(*args)
    Bases: _uniq, AnInfinity, InfinityElement
    Initialize self.
    
sqrt()
    EXAMPLES:

sage: (-oo).sqrt()
Traceback (most recent call last):
  ...  
SignError: cannot take square root of negative infinity
```

```python
definition

class sage.rings.infinity.PlusInfinity(*args)
    Bases: _uniq, AnInfinity, InfinityElement
    Initialize self.
    
sqrt()
    The square root of self.
    The square root of infinity is infinity.
    EXAMPLES:

sage: oo.sqrt()
+Infinity
```

```python
exception
definition

definition

class sage.rings.infinity.UnsignedInfinity(*args)
    Bases: _uniq, AnInfinity, InfinityElement
    Initialize self.
```

```python
class sage.rings.infinity.UnsignedInfinityRing_class
    Bases: Singleton, Ring
    Initialize self.
```
fraction_field()

The unsigned infinity ring isn’t an integral domain.

EXAMPLES:

```python
sage: UnsignedInfinityRing.fraction_field()
Traceback (most recent call last):
...  
TypeError: infinity 'ring' has no fraction field
```

gen(n=0)

The “generator” of self is the infinity object.

EXAMPLES:

```python
sage: UnsignedInfinityRing.gen()
Infinity
sage: UnsignedInfinityRing.gen(1)
Traceback (most recent call last):
...  
IndexError: UnsignedInfinityRing only has one generator
```

gens()

The “generator” of self is the infinity object.

EXAMPLES:

```python
sage: UnsignedInfinityRing.gens()
[Infinity]
```

less_than_infinity()

This is the element that represents a finite value.

EXAMPLES:

```python
sage: UnsignedInfinityRing.less_than_infinity()
A number less than infinity
sage: UnsignedInfinityRing(5) is UnsignedInfinityRing.less_than_infinity()
True
```

ngens()

The unsigned infinity ring has one “generator.”

EXAMPLES:

```python
sage: UnsignedInfinityRing.ngens()
1
sage: len(UnsignedInfinityRing.gens())
1
```

sage.rings.infinity.is_Infinite(x)

This is a type check for infinity elements.

EXAMPLES:
sage: sage.rings.infinity.is_Infinite(oo)
True
sage: sage.rings.infinity.is_Infinite(-oo)
True
sage: sage.rings.infinity.is_Infinite(unsigned_infinity)
True
sage: sage.rings.infinity.is_Infinite(3)
False
sage: sage.rings.infinity.is_Infinite(RR(infinity))
False
sage: sage.rings.infinity.is_Infinite(ZZ)
False

sage.rings.infinity.test_comparison(ring)
Check comparison with infinity

INPUT:
• ring – a sub-ring of the real numbers

OUTPUT:
Various attempts are made to generate elements of ring. An assertion is triggered if one of these elements does not compare correctly with plus/minus infinity.

EXAMPLES:

sage: from sage.rings.infinity import test_comparison
sage: rings = [ZZ, QQ, RDF]
sage: rings += [RR, RealField(200)]
# needs sage.rings.real_mpfr
sage: rings += [RLF, RIF]
# needs sage.rings.real_interval_field
sage: for R in rings:
 ....:     print('testing {}'.format(R))
 ....:     test_comparison(R)
testing Integer Ring
testing Rational Field
testing Real Double Field...
sage: test_comparison(AA) # needs sage.rings.number_field

Comparison with number fields does not work:

sage: x = polygen(ZZ, 'x')
sage: K.<sqrt3> = NumberField(x^2 - 3)
# needs sage.rings.number_field
sage: (-oo < 1 + sqrt3) and (1 + sqrt3 < oo) # known bug
False

The symbolic ring handles its own infinities, but answers False (meaning: cannot decide) already for some very elementary comparisons:

sage: test_comparison(SR) # known bug
# needs sage.symbolic

(continues on next page)
Traceback (most recent call last):
  ...  
AssertionError: testing -1000.0 in Symbolic Ring: id = ...

sage.rings.infinity.test_signedInfinity(pos_inf)

Test consistency of infinity representations.

There are different possible representations of infinity in Sage. These are all consistent with the infinity ring, that is, compare with infinity in the expected way. See also github issue #14045

INPUT:
  * pos_inf – a representation of positive infinity.

OUTPUT:

An assertion error is raised if the representation is not consistent with the infinity ring.

Check that github issue #14045 is fixed:

```python
sage: InfinityRing(float('+inf')) +Infinity
sage: InfinityRing(float('-inf')) -Infinity
sage: oo > float('+inf') False
sage: oo == float('+inf') True
```

EXAMPLES:

```python
sage: from sage.rings.infinity import test_signedInfinity
sage: test_signedInfinity(oo)
sage: test_signedInfinity(float('+inf'))
sage: test_signedInfinity(RLF(oo))  # needs sage.rings.real_interval_field
sage: test_signedInfinity(RIF(oo))  # needs sage.rings.real_interval_field
sage: test_signedInfinity(SR(oo))  # needs sage.symbolic
```
10.1 Derivations

Let $A$ be a ring and $B$ be an bimodule over $A$. A derivation $d : A \to B$ is an additive map that satisfies the Leibniz rule

$$d(xy) = xd(y) + d(x)y.$$ 

If $B$ is an algebra over $A$ and if we are given in addition a ring homomorphism $\theta : A \to B$, a twisted derivation with respect to $\theta$ (or a $\theta$-derivation) is an additive map $d : A \to B$ such that

$$d(xy) = \theta(x)d(y) + d(x)y.$$ 

When $\theta$ is the morphism defining the structure of $A$-algebra on $B$, a $\theta$-derivation is nothing but a derivation. In general, if $\iota : A \to B$ denotes the defining morphism above, one easily checks that $\theta - \iota$ is a $\theta$-derivation.

This file provides support for derivations and twisted derivations over commutative rings with values in algebras (i.e. we require that $B$ is a commutative $A$-algebra). In this case, the set of derivations (resp. $\theta$-derivations) is a module over $B$.

Given a ring $A$, the module of derivations over $A$ can be created as follows:

```sage
A.<x,y,z> = QQ[]
M = A.derivation_module()
M
```

The method `gens()` returns the generators of this module:

```sage
M.gens()
```

We can combine them in order to create all derivations:

```sage
d = 2*M.gen(0) + z*M.gen(1) + (x^2 + y^2)*M.gen(2)
d
```

and now play with them:
General Rings, Ideals, and Morphisms, Release 10.2

\[
\text{sage: } d(x + y + z) \\
x^2 + y^2 + z + 2
\]

\[
\text{sage: } P = A\text{.random_element()} \\
\text{sage: } Q = A\text{.random_element()} \\
\text{sage: } d(P*Q) == P*d(Q) + d(P)*Q \\
\text{True}
\]

Alternatively we can use the method \textit{derivation()} of the ring \( A \) to create derivations:

\[
\text{sage: } \text{Dx} = A\text{.derivation}(x); \text{Dx} \\
d/dx \\
\text{sage: } \text{Dy} = A\text{.derivation}(y); \text{Dy} \\
d/dy \\
\text{sage: } \text{Dz} = A\text{.derivation}(z); \text{Dz} \\
d/dz \\
\text{sage: } A\text{.derivation}([2, z, x^2+y^2]) \\
2 * d/dx + z * d/dy + (x^2 + y^2) * d/dz
\]

Sage knows moreover that \( M \) is a Lie algebra:

\[
\text{sage: } M\text{.category()} \\
\text{Join of } \\
\text{Category of lie algebras with basis over \text{Rational Field} and } \\
\text{Category of modules with basis over } \\
\text{Multivariate Polynomial Ring in x, y, z over \text{Rational Field}}
\]

Computations of Lie brackets are implemented as well:

\[
\text{sage: } \text{Dx.bracket(Dy)} \\
\emptyset \\
\text{sage: } d\text{.bracket(Dx)} \\
-2 * x * d/dz
\]

At the creation of a module of derivations, a codomain can be specified:

\[
\text{sage: } B = A\text{.fraction_field()} \\
\text{sage: } A\text{.derivation_module}(B) \\
\text{Module of derivations from Multivariate Polynomial Ring in x, y, z over \text{Rational Field} to Fraction Field of Multivariate Polynomial Ring in x, y, z over \text{Rational Field}}
\]

Alternatively, one can specify a morphism \( f \) with domain \( A \). In this case, the codomain of the derivations is the codomain of \( f \) but the latter is viewed as an algebra over \( A \) through the homomorphism \( f \). This construction is useful, for example, if we want to work with derivations on \( A \) at a certain point, e.g. \((0, 1, 2)\). Indeed, in order to achieve this, we first define the evaluation map at this point:

\[
\text{sage: } ev = A\text{.hom}([\text{QQ(0)}, \text{QQ(1)}, \text{QQ(2)}]) \\
\text{sage: } ev \\
\text{Ring morphism:} \\
\text{From: Multivariate Polynomial Ring in x, y, z over \text{Rational Field} } \\
\text{To: Rational Field} \\
\text{Defn: } x |--> 0 \\
\text{} y |--> 1 \\
\text{} z |--> 2
\]
Now we use this ring homomorphism to define a structure of $A$-algebra on $\mathbb{Q}$ and then build the following module of derivations:

```python
sage: M = A.derivation_module(ev)
sage: M
Module of derivations
from Multivariate Polynomial Ring in x, y, z over Rational Field
to Rational Field
sage: M.gens()
(d/dx, d/dy, d/dz)
```

Elements in $M$ then acts as derivations at $(0,1,2)$:

```python
sage: Dx = M.gen(0)
sage: Dy = M.gen(1)
sage: Dz = M.gen(2)
sage: f = x^2 + y^2 + z^2
sage: Dx(f)  # = 2*x evaluated at (0,1,2)
0
sage: Dy(f)  # = 2*y evaluated at (0,1,2)
2
sage: Dz(f)  # = 2*z evaluated at (0,1,2)
4
```

Twisted derivations are handled similarly:

```python
sage: theta = B.hom([B(y),B(z),B(x)])
sage: theta
Ring endomorphism of Fraction Field of
Multivariate Polynomial Ring in x, y, z over Rational Field
Defn: x |--> y
       y |--> z
       z |--> x
sage: M = B.derivation_module(twist=theta)
sage: M
Module of twisted derivations over Fraction Field of Multivariate Polynomial Ring
in x, y, z over Rational Field (twisting morphism: x |--> y, y |--> z, z |--> x)
```

Over a field, one proves that every $\theta$-derivation is a multiple of $\theta - id$, so that:

```python
sage: d = M.gen(); d
[x |--> y, y |--> z, z |--> x] - id
```

and then:

```python
sage: d(x)
-x + y
sage: d(y)
-y + z
sage: d(z)
x - z
sage: d(x + y + z)
0
```
class sage.rings.derivation.RingDerivation

An abstract class for twisted and untwisted derivations over commutative rings.

codomain()

Return the codomain of this derivation.

EXAMPLES:

```python
sage: R.<x> = QQ[]
sage: f = R.derivation(); f
d/dx
sage: f.codomain()
Univariate Polynomial Ring in x over Rational Field
sage: f.codomain() is R
True
```

domain()

Return the domain of this derivation.

EXAMPLES:

```python
sage: S.<y> = R[]
sage: M = R.derivation_module(S)
sage: M.random_element().codomain()
Univariate Polynomial Ring in y over
  Univariate Polynomial Ring in x over Rational Field
sage: M.random_element().codomain() is S
True
```

class sage.rings.derivation.RingDerivationModule(domain, codomain, twist=None)

A class for modules of derivations over a commutative ring.

basis()

Return a basis of this module of derivations.

EXAMPLES:

```python
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module()
sage: M.basis()
Family (d/dx, d/dy)
```
codomain()

Return the codomain of the derivations in this module.

EXAMPLES:

```
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module(); M
Module of derivations over Multivariate Polynomial Ring in x, y over Integer Ring
sage: M.codomain()
Multivariate Polynomial Ring in x, y over Integer Ring
```

defining_morphism()

Return the morphism defining the structure of algebra of the codomain over the domain.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: M = R.derivation_module()
sage: M.defining_morphism()
Identity endomorphism of Univariate Polynomial Ring in x over Rational Field
sage: S.<y> = R[]
sage: M = R.derivation_module(S)
sage: M.defining_morphism()
Polynomial base injection morphism:
  From: Univariate Polynomial Ring in x over Rational Field
  To:   Univariate Polynomial Ring in y over Univariate Polynomial Ring in x over Rational Field
sage: ev = R.hom([QQ(0)])
sage: M = R.derivation_module(ev)
sage: M.defining_morphism()
Ring morphism:
  From: Univariate Polynomial Ring in x over Rational Field
  To:   Rational Field
  Defn: x |--> 0
```

domain()

Return the domain of the derivations in this module.

EXAMPLES:

```
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module(); M
Module of derivations over Multivariate Polynomial Ring in x, y over Integer Ring
sage: M.domain()
Multivariate Polynomial Ring in x, y over Integer Ring
```

dual_basis()

Return the dual basis of the canonical basis of this module of derivations (which is that returned by the method basis()).

Note: The dual basis of \((d_1, \ldots, d_n)\) is a family \((x_1, \ldots, x_n)\) of elements in the domain such that \(d_i(x_j) = \delta_{ij}\) for \(i, j = 1, \ldots, n\).
1 and $d_i(x_j) = 0$ if $i \neq j$.

EXAMPLES:

```python
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module()
sage: M.basis()
Family (d/dx, d/dy)
sage: M.dual_basis()
Family (x, y)
```

gen($n=0$)

Return the $n$-th generator of this module of derivations.

INPUT:

- $n$ – an integer (default: 0)

EXAMPLES:

```python
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module(); M
Module of derivations over Multivariate Polynomial Ring in x, y over Integer Ring
sage: M.gen()
d/dx
sage: M.gen(1)
d/dy
```

gens()

Return the generators of this module of derivations.

EXAMPLES:

```python
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module(); M
Module of derivations over Multivariate Polynomial Ring in x, y over Integer Ring
sage: M.gens()
(d/dx, d/dy)
```

We check that, for a nontrivial twist over a field, the module of twisted derivation is a vector space of dimension 1 generated by $\text{twist} - \text{id}$:

```python
sage: K = R.fraction_field()
sage: theta = K.hom([K(y),K(x)])
sage: M = K.derivation_module(twist=theta); M
Module of twisted derivations over Fraction Field of Multivariate Polynomial Ring in x, y over Integer Ring (twisting morphism: \[x \mapsto y, y \mapsto x\])
sage: M.gens()
([x \mapsto y, y \mapsto x] - id,)
```

ngens()

Return the number of generators of this module of derivations.

EXAMPLES:
Indeed, generators are:

\[
\text{sage: } \text{M.gens()}
\]
\[
(\frac{\partial}{\partial x}, \frac{\partial}{\partial y})
\]

We check that, for a nontrivial twist over a field, the module of twisted derivation is a vector space of dimension 1 generated by \(\text{twist} - \text{id}\):

\[
\text{sage: } \text{K = R.fraction_field()}
\]
\[
\text{sage: } \text{theta = K.hom([K(y), K(x)])}
\]
\[
\text{sage: } \text{M = K.derivation_module(twist=theta); M}
\]
\[
\text{Module of twisted derivations over Fraction Field of Multivariate Polynomial Ring in x, y over Integer Ring (twisting morphism: x |--> y, y |--> x)}
\]
\[
\text{sage: } \text{M.ngens()}
\]
\[
1
\]
\[
\text{sage: } \text{M.gen()}
\]
\[
[x |--> y, y |--> x] - \text{id}
\]

\text{random_element(*args, **kwds)}

Return a random derivation in this module.

\text{ EXAMPLES: }

\[
\text{sage: } \text{R.<x,y> = ZZ[]}
\]
\[
\text{sage: } \text{M = R.derivation_module()}
\]
\[
\text{sage: } \text{M.random_element()} \quad \# \text{ random}
\]
\[
(x^2 + x^2y - 3y^2 + x + 1)\frac{\partial}{\partial x} + (-2x^2 + 3x^2y + 10y^2 + 2x + 8)\frac{\partial}{\partial y}
\]

\text{ring_of_constants()}

Return the subring of the domain consisting of elements \(x\) such that \(d(x) = 0\) for all derivation \(d\) in this module.

\text{ EXAMPLES: }

\[
\text{sage: } \text{R.<x,y> = QQ[]}
\]
\[
\text{sage: } \text{M = R.derivation_module()}
\]
\[
\text{sage: } \text{M.basis()}
\]
\[
\text{Family } (\frac{\partial}{\partial x}, \frac{\partial}{\partial y})
\]
\[
\text{sage: } \text{M.ring_of_constants()}
\]
\[
\text{Rational Field}
\]

\text{some_elements()}

Return a list of elements of this module.

\text{ EXAMPLES: }
twisting_morphism()

Return the twisting homomorphism of the derivations in this module.

EXAMPLES:

```python
sage: R.<x,y> = ZZ[]
sage: theta = R.hom([y,x])
sage: M = R.derivation_module(twist=theta); M
Module of twisted derivations over Multivariate Polynomial Ring in x, y
    over Integer Ring (twisting morphism: x |--> y, y |--> x)
sage: M.twisting_morphism()
Ring endomorphism of Multivariate Polynomial Ring in x, y over Integer Ring
    Defn: x |--> y
    y |--> x
```

When the derivations are untwisted, this method returns nothing:

```python
sage: M = R.derivation_module()
sage: M.twisting_morphism()

```

class sage.rings.derivation.RingDerivationWithTwist_generic(parent, scalar=0)

Bases: RingDerivation

The class handles $\theta$-derivations of the form $\lambda(\theta - \iota)$ (where $\iota$ is the defining morphism of the codomain over the domain) for a scalar $\lambda$ varying in the codomain.

extend_to_fraction_field()

Return the extension of this derivation to fraction fields of the domain and the codomain.

EXAMPLES:

```python
sage: R.<x,y> = ZZ[]
sage: theta = R.hom([y,x])
sage: d = R.derivation(x, twist=theta)
sage: d
x*([x |--> y, y |--> x] - id)
sage: D = d.extend_to_fraction_field(); D
# needs sage.libs.singular
x*([x |--> y, y |--> x] - id)
sage: D.domain()
# needs sage.libs.singular
Fraction Field of Multivariate Polynomial Ring in x, y over Integer Ring
sage: D(1/x)
# needs sage.libs.singular
(x - y)/y
```

list()

Return the list of coefficient of this twisted derivation on the canonical basis.
postcompose(morphism)

Return the twisted derivation obtained by applying first this twisted derivation and then morphism.

INPUT:

• morphism – a homomorphism of rings whose domain is the codomain of this derivation or a ring into which the codomain of this derivation

EXAMPLES:

\[
\text{sage: } R.<x,y> = \mathbb{Q}[x,y]
\]
\[
\text{sage: } K = R.fraction_field()
\]
\[
\text{sage: } \theta = K.hom([y,x])
\]
\[
\text{sage: } M = K.derivation_module(twist=\theta)
\]
\[
\text{sage: } M.basis()
\]
\[
(\text{twisting\_morphism} - \text{id}, )
\]
\[
\text{sage: } f = (x+y) \circ M.\text{gen}()
\]
\[
\text{sage: } f
\]
\[
(x + y) \circ (\text{twisting\_morphism} - \text{id})
\]
\[
\text{sage: } f.\text{list}()
\]
\[
[x + y]
\]

precompose(morphism)

Return the twisted derivation obtained by applying first morphism and then this twisted derivation.

INPUT:

• morphism – a homomorphism of rings whose codomain is the domain of this derivation or a ring that coerces to the domain of this derivation

EXAMPLES:

\[
\text{sage: } R.<x,y> = \mathbb{Z}[x,y]
\]
\[
\text{sage: } \theta = R.hom([y,x])
\]
\[
\text{sage: } D = R.derivation(x, twist=\theta); D
\]
\[
\begin{align*}
\text{x} & \rightarrow y, y \rightarrow x - \text{id} \\
\end{align*}
\]
\[
\text{sage: } f = R.hom([x^2, y^3])
\]
\[
\text{sage: } g = D.precompose(f); g
\]
\[
\begin{align*}
\text{x} & \rightarrow y^2, y \rightarrow x^3 - \text{x} \rightarrow x^2, y \rightarrow y^3 \\
\end{align*}
\]
Observe that the $g$ is no longer a $\theta$-derivation but a $(f \circ \theta)$-derivation:

```python
sage: g.parent().twisting_morphism()
Ring endomorphism of Multivariate Polynomial Ring in x, y over Integer Ring
  Defn: x |--> y^3
        y |--> x^2
```

class `sage.rings.derivation.RingDerivationWithoutTwist`

Bases: `RingDerivation`

An abstract class for untwisted derivations.

**extend_to_fraction_field()**

Return the extension of this derivation to fraction fields of the domain and the codomain.

**EXAMPLES:**

```python
sage: S.<x> = QQ[]
sage: d = S.derivation()
sage: d
d/dx
sage: D = d.extend_to_fraction_field()
sage: D
d/dx
sage: D.domain()
Fraction Field of Univariate Polynomial Ring in x over Rational Field
sage: D(1/x)
-1/x^2
```

**is_zero()**

Return True if this derivation is zero.

**EXAMPLES:**

```python
sage: R.<x,y> = ZZ[]
sage: f = R.derivation(); f
d/dx
sage: f.is_zero()
False
sage: (f-f).is_zero()
True
```

**list()**

Return the list of coefficient of this derivation on the canonical basis.

**EXAMPLES:**
sage: R.<x,y> = QQ[]
sage: M = R.derivation_module()
sage: M.basis()
Family (d/dx, d/dy)

sage: R.derivation(x).list()
[1, 0]
sage: R.derivation(y).list()
[0, 1]

sage: f = x*R.derivation(x) + y*R.derivation(y); f
x*d/dx + y*d/dy
sage: f.list()
[x, y]

\textbf{monomial\_coefficients()}

Return dictionary of nonzero coordinates (on the canonical basis) of this derivation.

More precisely, this returns a dictionary whose keys are indices of basis elements and whose values are the corresponding coefficients.

\textbf{EXAMPLES:}

sage: R.<x,y> = QQ[]
sage: M = R.derivation_module()
sage: M.basis()
Family (d/dx, d/dy)

sage: R.derivation(x).monomial_coefficients()
{0: 1}
sage: R.derivation(y).monomial_coefficients()
{1: 1}

sage: f = x*R.derivation(x) + y*R.derivation(y); f
x*d/dx + y*d/dy
sage: f.monomial_coefficients()
{0: x, 1: y}

\textbf{postcompose(morphism)}

Return the derivation obtained by applying first this derivation and then \texttt{morphism}.

\textbf{INPUT:}

\begin{itemize}
  \item \texttt{morphism} – a homomorphism of rings whose domain is the codomain of this derivation or a ring into which the codomain of this derivation coerces
\end{itemize}

\textbf{EXAMPLES:}

sage: A.<x,y>= QQ[]
sage: ev = A.hom([QQ(0), QQ(1)])
sage: Dx = A.derivation(x)
sage: Dy = A.derivation(y)

We can define the derivation at (0, 1) just by postcomposing with \texttt{ev}:
Note that we cannot avoid the creation of the evaluation morphism: if we pass in QQ instead, an error is raised since there is no coercion morphism from A to QQ:

```
sage: Dx.postcompose(QQ)
Traceback (most recent call last):
  ...
TypeError: the codomain of the derivation does not coerce to the given ring
```

Note that this method cannot be used to compose derivations:

```
sage: Dx.precompose(Dy)
Traceback (most recent call last):
  ...
TypeError: you must give an homomorphism of rings
```

`precompose(morphism)`

Return the derivation obtained by applying first `morphism` and then this derivation.

**INPUT:**

- `morphism` – a homomorphism of rings whose codomain is the domain of this derivation or a ring that coerces to the domain of this derivation

**EXAMPLES:**

```
sage: A.<x> = QQ[]
sage: B.<x,y> = QQ[]
sage: D = B.derivation(x) - 2*x*B.derivation(y); D
d/dx - 2*x*d/dy
```

When restricting to A, the term d/dy disappears (since it vanishes on A):

```
sage: D.precompose(A)
d/dx
```

If we restrict to another well chosen subring, the derivation vanishes:

```
sage: C.<t> = QQ[]
sage: f = C.hom([x^2 + y]); f
Ring morphism:
  From: Univariate Polynomial Ring in t over Rational Field
  To:  Multivariate Polynomial Ring in x, y over Rational Field
  Defn: t |--> x^2 + y
sage: D.precompose(f)
0
```

Note that this method cannot be used to compose derivations:
The `pth_power()` function returns the \( p \)-th power of this derivation where \( p \) is the characteristic of the domain.

**Note:** Leibniz rule implies that this is again a derivation.

**EXAMPLES:**

```python
sage: # needs sage.rings.finite_rings
sage: R.<x,y> = GF(5)[]
sage: Dx = R.derivation(x)
sage: Dx.pth_power()
0
sage: (x*Dx).pth_power()
x^d/dx
sage: (x^6*Dx).pth_power()
x^26^d/dx
sage: Dy = R.derivation(y)  # needs sage.rings.finite_rings
sage: (x*Dx + y*Dy).pth_power()  # needs sage.rings.finite_rings
x^d/dx + y^d/dy
```

An error is raised if the domain has characteristic zero:

```python
sage: R.<x,y> = QQ[
Traceback (most recent call last):
  ... TypeError: the domain of the derivation must have positive and prime
characteristic
```

or if the characteristic is not a prime number:

```python
sage: R.<x,y> = Integers(10)[
Traceback (most recent call last):
  ... TypeError: the domain of the derivation must have positive and prime
characteristic
```

**Class:** `sage.rings.derivation.RingDerivationWithoutTwist_fraction_field`

Bases: `RingDerivationWithoutTwist_wrapper`

This class handles derivations over fraction fields.
class sage.rings.derivation.RingDerivationWithoutTwist_function

Bases: RingDerivationWithoutTwist

A class for untwisted derivations over rings whose elements are either polynomials, rational fractions, power series or Laurent series.

is_zero()

Return True if this derivation is zero.

EXAMPLES:

```
sage: R.<x,y> = ZZ[]
sage: f = R.derivation(); f
d/dx
sage: f.is_zero()
False
sage: (f-f).is_zero()
True
```

list()

Return the list of coefficient of this derivation on the canonical basis.

EXAMPLES:

```
sage: R.<x,y> = GF(5)[[]]
sage: M = R.derivation_module()
sage: M.basis()
Family (d/dx, d/dy)
sage: R.derivation(x).list()
[1, 0]
sage: R.derivation(y).list()
[0, 1]
sage: f = x*R.derivation(x) + y*R.derivation(y); f
x*d/dx + y*d/dy
sage: f.list()
[x, y]
```

class sage.rings.derivation.RingDerivationWithoutTwist_quotient

Bases: RingDerivationWithoutTwist_wrapper

This class handles derivations over quotient rings.

class sage.rings.derivation.RingDerivationWithoutTwist_wrapper

Bases: RingDerivationWithoutTwist

This class is a wrapper for derivation.

It is useful for changing the parent without changing the computation rules for derivations. It is used for derivations over fraction fields and quotient rings.

list()

Return the list of coefficient of this derivation on the canonical basis.

EXAMPLES:
```python
sage: # needs sage.libs.singular
sage: R.<X,Y> = GF(5)[]
```
```python
sage: S.<x,y> = R.quo([X^5, Y^5])
```
```python
sage: M = S.derivation_module()
```
```python
sage: M.basis()
Family (d/dx, d/dy)
```
```python
sage: S.derivation(x).list()
[1, 0]
```
```python
sage: S.derivation(y).list()
[0, 1]
```
```python
sage: f = x*S.derivation(x) + y*S.derivation(y); f
x*d/dx + y*d/dy
```
```python
sage: f.list()
[x, y]
```

**class** `sage.rings.derivation.RingDerivationWithoutTwist_zero(parent, arg=None)`

**Bases:** `RingDerivationWithoutTwist`

This class can only represent the zero derivation.

It is used when the parent is the zero derivation module (e.g., when its domain is ZZ, QQ, a finite field, etc.)

**is_zero()**

Return True if this derivation vanishes.

**EXAMPLES:**

```python
sage: M = QQ.derivation_module()
sage: M().is_zero()
True
```

**list()**

Return the list of coefficient of this derivation on the canonical basis.

**EXAMPLES:**

```python
sage: M = QQ.derivation_module()
sage: M().list()
[]
```
INDICES AND TABLES

• Index
• Module Index
• Search Page
sage.rings.abc, 25
sage.rings.big_oh, 159
sage.rings.derivation, 171
sage.rings.fraction_field, 95
sage.rings.fraction_field_element, 101
sage.rings.generic, 155
sage.rings.homset, 74
sage.rings.ideal, 35
sage.rings.ideal_monoid, 51
sage.rings.infinity, 161
sage.rings.localization, 107
sage.rings.morphism, 55
sage.rings.noncommutative_ideals, 52
sage.rings.numbers_abc, 169
sage.rings.quotient_ring, 77
sage.rings.quotient_ring_element, 90
sage.rings.ring, 1
sage.rings.ring_extension, 117
sage.rings.ring_extension_element, 139
sage.rings.ring_extension_morphism, 150
INDEX

A

absolute_base() (sage.rings.ring_extension.RingExtension_generic method), 125
absolute_degree() (sage.rings.ring_extension.RingExtension_generic method), 125
absolute_norm() (sage.rings.ideal.Ideal_generic method), 38
additive_order() (sage.rings.ring_extension_element.RingExtensionElement method), 139
Algebra (class in sage.rings.ring), 2
algebraic_closure() (sage.rings.ring.Field method), 10
AlgebraicField (class in sage.rings.abc), 25
AlgebraicField_common (class in sage.rings.abc), 25
AlgebraicRealField (class in sage.rings.abc), 26
ambient() (sage.rings.quotient_ring.QuotientRing_nc method), 82
AnInfinity (class in sage.rings.infinity), 163
apply_morphism() (sage.rings.ideal.Ideal_generic method), 38
associated_primes() (sage.rings.ideal.Ideal_generic method), 39

B

backend() (sage.rings.ring_extension.RingExtension_generic method), 125
backend() (sage.rings.ring_extension_element.RingExtensionElement method), 139
base() (sage.rings.ring_extension.RingExtension_generic method), 126
base_extend() (sage.rings.ring.Ring method), 16
base_map() (sage.rings.morphism.RingHomomorphism_injective method), 72
base_map() (sage.rings.ring_extension_morphism.RingExtensionHomomorphism method), 151
base_ring() (sage.rings.fraction_field.FractionField_generic method), 98
base_ring() (sage.rings.ideal.Ideal_generic method), 39
bases() (sage.rings.ring_extension.RingExtension_generic method), 126
basis() (sage.rings.derivation.RingDerivationModule method), 174
basis_over() (sage.rings.ring_extension.RingExtensionWithBasis method), 120

C

CallableSymbolicExpressionRing (class in sage.rings.abc), 26
category() (sage.rings.ideal.Ideal_generic method), 40
category() (sage.rings.ring.Ring method), 16
characteristic() (sage.rings.fraction_field.FractionField_generic method), 98
characteristic() (sage.rings.localization.Localization method), 112
characteristic() (sage.rings.quotient_ring.QuotientRing_nc method), 82
characteristic() (sage.rings.ring.Algebra method), 2
characteristic() (sage.rings.ring_extension.RingExtension_generic method), 127
charpoly() (sage.rings.ring_extension_element.RingExtensionWithBasisElement method), 127
class_group() (sage.rings.ring.PrincipalIdealDomain method), 14
class_number() (sage.rings.fraction_field.FractionField_1poly_field method), 97
codomain() (sage.rings.derivation.RingDerivation method), 174
codomain() (sage.rings.derivation.RingDerivationModule method), 174
common_base() (in module sage.rings.ring_extension), 137
CommutativeAlgebra (class in sage.rings.ring), 2
CommutativeRing (class in sage.rings.ring), 3
ComplexBallField (class in sage.rings.abc), 27
ComplexDoubleField (class in sage.rings.abc), 27
ComplexField (class in sage.rings.abc), 27
ComplexIntervalField (class in sage.rings.abc), 28
construction() (sage.rings.fraction_field.FractionField_generic method), 98
construction() (sage.rings.quotient_ring.QuotientRing_nc method), 83
construction() (sage.rings.ring_extension.RingExtensionElement method), 127
content() (sage.rings.ring.PrincipalIdealDomain method), 14
cover() (sage.rings.quotient_ring.QuotientRing_nc method), 83
cover_ring() (sage.rings.quotient_ring.QuotientRing_nc method), 84
create_key_and_extra_args()
  (sage.rings.ring_extension.RingExtensionFactory method), 119
create_object() (sage.rings.ring_extension.RingExtensionFactory method), 119
Cyclic() (in module sage.rings.ideal), 35

d
DedekindDomain (class in sage.rings.ring), 8
defining_ideal() (sage.rings.quotient_ring.QuotientRing_nc method), 84
defining_morphism()
  (sage.rings.derivation.RingDerivationModuleGenerator method), 175
defining_morphism()
  (sage.rings.ring_extension.RingExtension_generic method), 127
degree() (sage.rings.ring_extension.RingExtension generic method), 128
degree_over() (sage.rings.ring_extension.RingExtension_generic method), 129
denominator()
  (sage.rings.fraction_field_element.FractionFieldElement method), 101
denominator()
  (sage.rings.localization.LocalizationElement method), 114
denominator()
  (sage.rings.ring_extension_element.RingExtensionElement method), 142
derivation()
  (sage.rings.ring.CommutativeRing method), 3
derivation_module()
  (sage.rings.ring.CommutativeRing method), 4
divides()
  (sage.rings.ideal.Ideal_principal method), 48
divides()
  (sage.rings.ring.Field method), 10
domain()
  (sage.rings.derivation.RingDerivation method), 174
domain()
  (sage.rings.derivation.RingDerivationModuleGenerator method), 175
dual_basis()
  (sage.rings.derivation.RingDerivationModuleGenerator method), 175

e
Element
  (sage.rings.homset.RingHomset_generic attribute), 74
Element
  (sage.rings.homset.RingHomset_quo_ring attribute), 76

E
Element
  (sage.rings.homset.RingHomset_generic attribute), 74
FractionField
  (class in sage.rings.fraction_field), 97
FractionField_generic
  (class in sage.rings.fraction_field), 98
FractionFieldElement
  (class in sage.rings.fraction_field_element), 101
General Rings, Ideals, and Morphisms, Release 10.2

FractionFieldElement_lpoly_field (class in sage.rings.fraction_field_element), 104

FractionFieldEmbedding (class in sage.rings.fraction_field), 96

FractionFieldEmbeddingSection (class in sage.rings.fraction_field), 97

free_module() (sage.rings.ring_extension.RingExtensionWithBasis method), 121

free_resolution() (sage.rings.ideal.Ideal_generic method), 40

frobenius_endomorphism() (sage.rings.ring.CommutativeRing method), 6

FrobeniusEndomorphism_generic (class in sage.rings.morphism), 61

from_base_ring() (sage.rings.ring_extension.RingExtension_generic method), 130

function_field() (sage.rings.fraction_field.FractionField_element method), 97

G

gcd() (sage.rings.ideal.Ideal_pid method), 46

gcd() (sage.rings.ring.PrincipalIdealDomain method), 15

gen() (sage.rings.derivation.RingDerivationModule method), 176

gen() (sage.rings.fraction_field.FractionField_generic method), 99

gen() (sage.rings.ideal.Ideal_generic method), 40

gen() (sage.rings.ideal.Ideal_principal method), 49

gen() (sage.rings.infinity.InfinityRing_class method), 164

gen() (sage.rings.infinity.UnsignedInfinityRing_class method), 167

gen() (sage.rings.localization.Localization method), 112

gen() (sage.rings.quotient_ring.quotientRing_quotientRing_nc method), 84

gen() (sage.rings.ring_extension.RingExtension_generic method), 131

generators() (in module sage.rings.ring_extension), 138

gens() (sage.rings.derivation.RingDerivationModule method), 176

gens() (sage.rings.ideal.Ideal_generic method), 41

gen() (sage.rings.infinity.InfinityRing_class method), 165

gen() (sage.rings.infinity.UnsignedInfinityRing_class method), 167

gen() (sage.rings.localization.Localization method), 112

gen() (sage.rings.ring_extension.RingExtension_generic method), 131

gen() (sage.rings.ring_extension.RingExtensionWithGen method), 124

gens_reduced() (sage.rings.ideal.IdealGeneric method), 41

graded_free_resolution() (sage.rings.ideal.Ideal_generic method), 41

has_coerce_map_from() (sage.rings.homset.RingHomset_generic method), 74

has_standard_involution() (sage.rings.ring.Algebra method), 2

hom() (sage.rings.ring_extension.RingExtension_generic method), 132

ideal() (sage.rings.quotient_ring.QuotientRing_nc method), 85

ideal() (sage.rings.ring.Field method), 11

Ideal_fractional (class in sage.rings.ideal), 38

Ideal_generic (class in sage.rings.ideal), 38

ideal_monoid() (sage.rings.ring.CommutativeRing method), 7

ideal_monoid() (sage.rings.ring.Ring method), 18

Ideal_nc (class in sage.rings.noncommutative_ideals), 52

Ideal_pid (class in sage.rings.ideal), 46

Ideal_principal (class in sage.rings.ideal), 48

IdealMonoid() (in module sage.rings.ideal_monoid), 51

IdealMonoid_c (class in sage.rings.ideal_monoid), 51

IdealMonoid_nc (class in sage.rings.noncommutative_ideals), 52

im_gens() (sage.rings.morphism.RingHomomorphism_im_gens method), 73

in_base() (sage.rings.ring_extension_element.RingExtensionElement method), 140

InfinityRing_class (class in sage.rings.infinity), 164

IntegerModRing (class in sage.rings.abc), 28

integral_closure() (sage.rings.ring.DedekindDomain method), 8

integral_closure() (sage.rings.ring.Field method), 11

IntegralDomain (class in sage.rings.ring), 12

inverse() (sage.rings.morphism.RingHomomorphism method), 61

inverse() (sage.rings.morphism.RingHomomorphism_from_base method), 70

inverse() (sage.rings.morphism.RingHomomorphism_from_fraction_field method), 71

inverse_image() (sage.rings.morphism.RingHomomorphism method), 64
inverse_of_unit() (sage.rings.localization.LocalizationElement method), 114
is_commutative() (sage.rings.infinity.InfinityRing_class method), 165
is_commutative() (sage.rings.quotient_ring.QuotientRing method), 85
is_injective() (sage.rings.ring_extension_morphism.MapFreeModuleToRelativeRing method), 150
is_integral() (sage.rings.ring_extension_morphism.MapRelativeRingToFreeModule method), 134
is_invertible() (sage.rings.morphism.RingHomomorphism method), 66
is_integer() (sage.rings.ring_extension_element.RingExtensionElement method), 41
is_integral_domain() (sage.rings.ring.IntegralDomain method), 9
is_integral() (sage.rings.ring_extension_element.RingExtensionElement method), 76
is_ideal() (sage.rings.ring_extension_element.RingExtensionElement method), 76
is_noetherian() (sage.rings.quotient_ring.QuotientRing method), 101
is_principal() (sage.rings.ideal.Ideal_generic method), 42
is_prime() (sage.rings.ideal.Ideal_generic method), 43
is_prime() (sage.rings.ideal.Ideal_pid method), 47
is_permutation() (sage.rings.fraction_field_element.FractionFieldElement method), 113
is_integral() (sage.rings.ring.IntegralDomain method), 9
is_prime() (sage.rings.ideal.Ideal_generic method), 76
is_prime() (sage.rings.ideal.Ideal pid method), 47
is_prime() (sage.rings.ideal.PrincipalIdealDomain method), 16
is_unit() (sage.rings.fraction_field_element.FractionFieldElement method), 113
is_prime() (sage.rings.ideal.Ideal_generic method), 152
is_principal() (sage.rings.ideal.Ideal_principal method), 43
is_prime_field() (sage.rings.ring.Field method), 21
is_principal() (sage.rings.ideal.Ideal_principal method), 43
is_quotient() (sage.rings.quotient_ring.QuotientRing method), 24
is_square() (sage.rings.fraction_field_element.FractionFieldElement method), 96
is_surjective() (sage.rings.morphism.RingHomomorphism method), 66
is_surjective() (sage.rings.ring_extension_morphism.MapToRelative method), 150
is_surjective() (sage.rings.ring_extension_morphism.MapToRelative method), 151
is_trivial() (sage.rings.ideal.Ideal_generic method), 44
is_unit() (sage.rings.localization.LocalizationElement method), 115
is_unit() (sage.rings.quotient_ring_element.QuotientRingElement method), 91
is_unit() (sage.rings.quotient_ring_element.QuotientRingElement method), 141
is_zero() (sage.rings.derivation.RingDerivationWithoutTwist method), 180
is_zero() (sage.rings.derivation.RingDerivationWithoutTwist function method), 184
is_zero() (sage.rings.derivation.RingDerivationWithoutTwist function method), 185
is_zero() (sage.rings.ring_extension_element.RingExtensionWithBasis method), 113
is_zero() (sage.rings.fraction_field_element.Element method), 102
is_zero() (sage.rings.infinity.InfinityRing_class method), 165
K
Katsura() (in module sage.rings.ideal), 50
kernel() (sage.rings.morphism.RingHomomorphism method), 66
kernel() (sage.rings.morphism.RingHomomorphism cover method), 68
krull_dimension() (sage.rings.localization.Localization method), 7
krull_dimension() (sage.rings.ring.CommutativeRing method), 9
krull_dimension() (sage.rings.ring.DedekindDomain method), 113
krull_dimension() (sage.rings.ring.Field method), 12
L
lc() (sage.rings.quotient_ring_element.QuotientRingElement method), 91
lcm() (sage.rings.infinity.AnInfinity method), 163
leaves() (sage.rings.generic.ProductTree method), 156
less_than_infinity() (sage.rings.infinity.UnsignedInfinityRing_class method), 167
LessThanInfinity (class in sage.rings.infinity), 165
lift() (sage.rings.morphism.RingHomomorphism method), 67
lift() (sage.rings.quotient_ring_element.QuotientRing_nc method), 87
make_element() (in module sage.rings.fraction_field_element), 105
make_element_old() (in module sage.rings.fraction_field_element), 105
MapFreeModuleToRelativeRing (class in sage.rings.ring.CommutativeRing), 150
MapRelativeRingToFreeModule (class in sage.rings.ring_extension_morphism), 151
maximal_order() (sage.rings.fraction_field.FractionField_Ipoly_field method), 98
minimal-associated_primes() (sage.rings.ideal.Ideal_generic method), 44
minpoly() (sage.rings.ring_extension_element.RingExtensionWithBasis method), 145
MinusInfinity (class in sage.rings.infinity), 166

Index
sage.rings.localization, 107
sage.rings.morphism, 55
sage.rings.noncommutative_ideals, 52
sage.rings.numbers_abc, 169
sage.rings.quotient_ring, 77
sage.rings.quotient_ring_element, 90
sage.rings, 1
sage.rings.ring_extension, 117
sage.rings.ring_extension_element, 139
sage.rings.ring_extension_morphism, 150
modulus() (sage.rings.ring_extension.RingExtensionWithGen
method), 124
monomial_coefficients()
(sage.rings.derivation.RingDerivationWithoutTwist
method), 181
monomials() (sage.rings.quotient_ring_element.QuotientRing
method), 92
morphism_from_cover()
(sage.rings.morphism.RingHomomorphism_from
method), 72
multiplicative_order()
(sage.rings.ring_extension_element.RingExtension
method), 141

N
natural_map() (sage.rings.homset.RingHomset_generic
method), 75
ngens() (sage.rings.derivation.RingDerivationModule
method), 176
ngens() (sage.rings.fraction_field.FractionField_generic
method), 99
ngens() (sage.rings.ideal.Ideal_generic method), 44
ngens() (sage.rings.infinity.InfinityRing_class
method), 165
ngens() (sage.rings.infinityUnsignedInfinityRing_class
method), 167
ngens() (sage.rings.localization.Localization method), 113
ngens() (sage.rings.quotient_ring.QuotientRing_nc
method), 88
ngens() (sage.rings.ring_extension.RingExtension_generic
method), 135
NoetherianRing (class in sage.rings.ring), 13
norm() (sage.rings.ideal.Ideal_generic method), 44
norm() (sage.rings.ring_extension_element.RingExtension
method), 146
normalize_extra_units() (in module
sage.rings.localization), 115
nth_root() (sage.rings.fraction_field.FractionField_element
method), 103
NumberField_cyclotomic (class in sage.rings.abc), 28
NumberField_quadratic (class in sage.rings.abc), 29
numerator() (sage.rings.fraction_field.FractionField_element
method), 103
numerator() (sage.rings.localization.LocalizationElement
method), 115
numerator() (sage.rings.ring_extension_element.RingExtensionFractionFieldElement
method), 143
O
O (in module sage.rings.big_oh), 159
one() (sage.rings.ring.Ring method), 21
Order (class in sage.rings.abc), 29
order() (sage.rings.ring.Ring method), 22
P
pAdicField (class in sage.rings.abc), 32
pAdicRing (class in sage.rings.abc), 33
parameter() (sage.rings.ring.EuclideanDomain
RingElement method), 10
PlusInfinity (class in sage.rings.infinity), 166
polynomial() (sage.rings.ring_extension_element.RingExtensionWithBase
method), 147
postcompose() (sage.rings.derivation.RingDerivationWithoutTwist
method), 181
postcompose() (sage.rings.derivation.RingDerivationWithTwist_generic
method), 179
power() (sage.rings.morphism.FrobeniusEndomorphism_generic
method), 61
precompose() (sage.rings.derivation.RingDerivationWithoutTwist
method), 182
precompose() (sage.rings.derivation.RingDerivationWithTwist_generic
method), 179
primary_decomposition()
(sage.rings.ideal.Ideal_generic method), 45
prime_subfield() (sage.rings.ring.Field method), 12
principal_ideal() (sage.rings.ring.Ring method), 22
PrincipalIdealDomain (class in sage.rings.ring), 14
print_options() (sage.rings.ring_extension.RingExtension_generic
method), 135
prod_with_derivative() (in module
sage.rings.generic), 157
ProductTree (class in sage.rings.generic), 155
pth_power() (sage.rings.derivation.RingDerivationWithoutTwist
method), 183
pushforward() (sage.rings.morphism.RingHomomorphism
method), 67
Python Enhancement Proposals
PEP 3141, 169
Q
quotientRing() (in module sage.rings.quotient_ring), 78
QuotientRing_generic (class in
sage.rings.quotient_ring), 81
QuotientRing_nc (class in sage.rings.quotient_ring), 81

196  Index
S
sage.rings.abc
module, 25
sage.rings.big_oh
module, 159
sage.rings.derivation
module, 171
sage.rings.fraction_field
module, 95
sage.rings.fraction_field_element
module, 101
sage.rings.generic
module, 155
sage.rings.homset
module, 74
sage.rings.ideal
module, 35
sage.rings.ideal_monoid
module, 51
sage.rings.infinity
module, 161
sage.rings.localization
module, 107
sage.rings.morphism
module, 55
sage.rings.noncommutative.ideals
module, 52
sage.rings.numbers_abcsage.rings.numbers_abc
module, 169
sage.rings.quotient_ring
module, 77
sage.rings.quotient_ring_element
module, 90
sage.rings.ring
module, 1
sage.rings.ring_extension
module, 117
sage.rings.ring_extension_element
module, 139
sage.rings.ring_extension_morphism
module, 150
section() (sage.rings.fraction_field.FractionFieldEmbedding
method), 97
side() (sage.rings.noncommutative_ideas.Ideal_nc
method), 53
sign() (sage.rings.infinity.FiniteNumber method), 164
sign() (sage.rings.infinity.LessThanInfinity method), 165
SymbolicRing (class in sage.rings.abc), 31
specialization() (sage.rings.fraction_field_element.FractionFieldElement
method), 104
sqrt() (sage.rings.infinity.FiniteNumber method), 164
sqrt() (sage.rings.infinity.MinusInfinity method), 166
sqrt() (sage.rings.infinity.PlusInfinity method), 166
sqrt() (sage.rings.ring_extension_element.RingExtensionElement
method), 142
support() (sage.rings.fraction_field_element.FractionFieldElement_Ipoly
method), 105
T
term_order() (sage.rings.quotient_ring.QuotientRing_nc
method), 89
test_comparison() (in module sage.rings.infinity), 168
test_signed_infinity() (in module sage.rings.infinity), 169
tower_bases() (in module sage.rings.ring_extension),
trace() (sage.rings.ring_extension_element.RingExtensionWithBasisElement
method), 148
twisting_morphism()
sage.rings.derivation.RingDerivationModule
method), 178
U
underlying_map() (sage.rings.morphism.RingHomomorphism_from_base
method), 70
unit_ideal() (sage.rings.ring.Ring method), 22
UniversalCyclotomicField (class in sage.rings.abc),
UnsignedInfinity (class in sage.rings.infinity), 166
UnsignedInfinityRing_class (class in
sage.rings.infinity), 166
V
valuation() (sage.rings.fraction_field_element.FractionFieldElement
method), 104
variable_names() (in module sage.rings.ring_extension), 139
variables() (sage.rings.quotient_ring_element.QuotientRingElement
method), 93
vector() (sage.rings.ring_extension_element.RingExtensionWithBasisElement
method), 149
Z
zero() (sage.rings.homset.RingHomset_generic
method), 75
zero() (sage.rings.ring.Ring method), 22
zero_ideal() (sage.rings.ring.Ring method), 23
zeta() (sage.rings.ring.Ring method), 23
zeta_order() (sage.rings.ring.Ring method), 24

198 Index