CONTENTS

1 Base Classes for Rings, Algebras and Fields 1
2 Ideals 35
3 Ring Morphisms 55
4 Quotient Rings 79
5 Fraction Fields 99
6 Localization 111
7 Ring Extensions 123
8 Generic Data Structures and Algorithms for Rings 159
9 Utilities 163
10 Derivation 175
11 Indices and Tables 191
Python Module Index 193
Index 195
1.1 Rings

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

\begin{itemize}
\item \texttt{Ring}
\item \texttt{Algebra}
\item \texttt{CommutativeRing}
\item \texttt{NoetherianRing}
\item \texttt{CommutativeAlgebra}
\item \texttt{IntegralDomain}
\item \texttt{DedekindDomain}
\item \texttt{PrincipalIdealDomain}
\end{itemize}

Subclasses of \texttt{PrincipalIdealDomain} are

\begin{itemize}
\item \texttt{EuclideanDomain}
\item \texttt{Field}
\item \texttt{FiniteField}
\end{itemize}

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, \texttt{Field} cannot be a subclass of both \texttt{NoetherianRing} and \texttt{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 \texttt{DedekindDomain} class.)

AUTHORS:

\begin{itemize}
\item David Harvey (2006-10-16): changed \texttt{CommutativeAlgebra} to derive from \texttt{CommutativeRing} instead of from \texttt{Algebra}.
\end{itemize}
class sage.rings.ring.Algebra

Bases: Ring

Generic algebra

class attribute for additional examples. Here are some examples that explicitly use the Algebra class.

EXAMPLES:

```python
sage: A = Algebra(ZZ); A
<....>
```

```python
sage: A.characteristic()
```

```python
sage: A = Algebra(GF(7^3, 'a'))
```

```python
sage: A.characteristic()
```

```python
sage: B = QuaternionAlgebra(2)
```

```python
sage: B.has_standard_involution()
```

```python
sage: R.<x> = PolynomialRing(QQ)
```

```python
sage: K.<u> = NumberField(x**2 - 2)
```

```python
sage: A = QuaternionAlgebra(K, -2, 5)
```

```python
sage: A.has_standard_involution()
```

```python
sage: L.<a,b> = FreeAlgebra(QQ, 2)
```

```python
sage: L.has_standard_involution()
```

Traceback (most recent call last):
... NotImplementedError: 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()                      # optional - sage.modules
d/dx

    In that case, arg could be a generator:

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

    or a list of coefficients:
It is not possible to define derivations with respect to a polynomial which is not a variable:

```python
sage: R.derivation(x^2)
```

Traceback (most recent call last):
... ValueError: unable to create the derivation

Here is an example with twisted derivations:

```python
sage: R.<x,y,z> = QQ[]
sage: theta = R.hom([x^2, y^2, z^2])
sage: f = R.derivation(twist=theta); f
```

```python
sage: f.parent()
```

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 \( \theta - \text{id} \):

```python
sage: R.derivation(1, twist=theta)
```

```python
sage: R.derivation(x, twist=theta)
```

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:

\[
d(xy) = \theta(x)d(y) + d(x)y.
\]
sage: R.<x,y,z> = QQ[]
sage: M = R.derivation_module(); M
# optional - sage.modules
Module of derivations over
Multivariate Polynomial Ring in x, y, z over Rational Field
sage: M.gens()
# optional - sage.modules
(d/dx, d/dy, d/dz)

We can specify a different codomain:

sage: K = R.fraction_field()
sage: M = R.derivation_module(K); M
# optional - sage.modules
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
# optional - sage.modules
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)
# optional - sage.modules
sage: M
# optional - sage.modules
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
# optional - sage.modules
d/dx
sage: Dy = M.gen(1); Dy
# optional - sage.modules
d/dy
sage: Dz = M.gen(2); Dz
# optional - sage.modules
d/dz
sage: f = x^2 + y^2 + z^2
sage: Dx(f)  # = 2x evaluated at (0,1,2)
(continues on next page)
optional - sage.modules

sage: Dy(f)  # = 2*y evaluated at (0,1,2)  
optional - sage.modules

sage: Dz(f)  # = 2*z evaluated at (0,1,2)  
optional - sage.modules

An example with a twisting homomorphism:

```python
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:

```python
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

```python
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
**fraction_field()**
Return the fraction field of self.

EXAMPLES:

```python
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
```

**frobenius_endomorphism(n=1)**

INPUT:
- `n` – a nonnegative integer (default: 1)

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

EXAMPLES:

```python
sage: K.<u> = PowerSeriesRing(GF(5))
# optional - sage.rings.finite_rings
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)
# optional - sage.rings.finite_rings
u^5
```

We can specify a power:

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

**ideal_monoid()**

Return the monoid of ideals of this ring.

EXAMPLES:

```python
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:
```python
sage: QQ.is_commutative()
True
sage: ZpCA(7).is_commutative()
# optional - sage.rings.padics
True
sage: A = QuaternionAlgebra(QQ, -1, -3, names=('i', 'j', 'k')); A
Quaternion Algebra (-1, -3) with base ring Rational Field
sage: A.is_commutative()
# optional - sage.combinat sage.modules
False
```

### 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(*)

Return the localization of self at the given additional units.

**EXAMPLES:**

```python
sage: R.<x, y> = GF(3)[]
# optional - sage.rings.finite_rings
sage: R.localization((x*y, x**2 + y**2))
# optional - 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 _   # optional - sage.rings.finite_rings
True
```

### 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: x = polygen(ZZ, 'x')
sage: K = NumberField(x**2 + 1, 's')    # optional - sage.rings.number_field
sage: OK = K.ring_of_integers()          # optional - sage.rings.number_field
sage: OK.integral_closure()              # optional - sage.rings.number_field
Gaussian Integers in Number Field in s
```

(continues on next page)
with defining polynomial \( x^2 + 1 \)

```python
sage: OK.integral_closure() == OK
#optional - sage.rings.number_field
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')
#optional - sage.rings.number_field
sage: OK = K.ring_of_integers()
#optional - sage.rings.number_field
sage: OK.is_integrally_closed()
#optional - sage.rings.number_field
True
```

These, however, are not Dedekind domains:

```python
sage: QQ.is_integrally_closed()
True
sage: S = ZZ[\sqrt{5}]; S.is_integrally_closed()
#optional - 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:

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

(continues on next page)
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.krull_dimension()
1
sage: x = polygen(ZZ, 'x')
```

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

```
sage: OK = K.ring_of_integers()
```

```
sage: OK.krull_dimension()
1
```

The following are not Dedekind domains but have a krull_dimension function:

```python
sage: QQ.krull_dimension()
0
```

```python
sage: T.<x,y> = PolynomialRing(QQ,2); T
Multivariate Polynomial Ring in x, y over Rational Field
```

```python
sage: T.krull_dimension()
2
```

```python
sage: U.<x,y,z> = PolynomialRing(ZZ,3); U
Multivariate Polynomial Ring in x, y, z over Integer Ring
```

```python
sage: U.krull_dimension()
4
```

```python
sage: K.<i> = QuadraticField(-1)
```

```python
sage: R = K.order(2*i); R
Order in Number Field in i with defining polynomial x^2 + 1 with i = 1*I
```

```python
sage: R.is_maximal()
False
```

```python
sage: R.krull_dimension()
1
```

```python
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:**

```
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:**

```
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()
Real Field with 53 bits of precision
sage: CC.fraction_field()
Complex Field with 53 bits of precision
```

(continues on next page)
General Rings, Ideals, and Morphisms, Release 10.1

\[
\text{sage: } x = \text{polygen}(\mathbb{Z}, 'x') \\
\text{sage: } F = \text{NumberField}(x^2 + 1, 'i') \\
\quad \text{# optional - sage.rings.number_field} \\
\text{sage: } F.\text{fraction_field()} \\
\quad \text{# optional - sage.rings.number_field} \\
\begin{align*}
\text{Number Field in } i \text{ with defining polynomial } x^2 + 1
\end{align*}
\]

\text{ideal}(\text{**gens, **kwds})

Return the ideal generated by gens.

EXAMPLES:

\[
\text{sage: } \text{QQ.ideal}(2) \\
\text{Principal ideal (1) of Rational Field} \\
\text{sage: } \text{QQ.ideal}(0) \\
\text{Principal ideal (0) of Rational Field}
\]

\text{integral_closure}()

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

EXAMPLES:

\[
\text{sage: } \text{QQ.integral_closure()} \\
\text{Rational Field} \\
\text{sage: } \text{Frac(ZZ['x,y']).integral_closure()} \\
\text{Fraction Field of Multivariate Polynomial Ring in x, y over Integer Ring}
\]

\text{is_field}(\text{proof=True})

Return True since this is a field.

EXAMPLES:

\[
\text{sage: } \text{Frac(ZZ['x,y']).is_field()} \\
\text{True}
\]

\text{is_integrally_closed}()

Return True since fields are trivially integrally closed in their fraction field (since they are their own fraction field).

EXAMPLES:

\[
\text{sage: } \text{Frac(ZZ['x,y']).is_integrally_closed()} \\
\text{True}
\]

\text{is_noetherian}()

Return True since fields are Noetherian rings.

EXAMPLES:

\[
\text{sage: } \text{QQ.is_noetherian()} \\
\text{True}
\]

\text{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')  
  # optional - sage.rings.finite_rings
sage: k.prime_subfield()  
  # optional - 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()  
  # optional - sage.rings.finite_rings
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

1.1. Rings
**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:**

```python
sage: ZZ.is_integrally_closed()
True
sage: QQ.is_integrally_closed()
True
sage: QQbar.is_integrally_closed()  # optional - sage.rings.number_field
True
sage: GF(5).is_integrally_closed()  # optional - sage.rings.finite_rings
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.

**EXAMPLES:**

```python
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.

EXAMPLES:
sage: QQ.class_group()
Optional - 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.

EXAMPLES:
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

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
13 * 17 * 19^-1 * 23^-1
7^-1 * 11^-1 * 19^-1 * 23^-1

Note the changes to the second entry:
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
13 * 17 * 19^-1 * 23^-1
7^-1 * 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
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:

\begin{verbatim}
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
\end{verbatim}

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:

\begin{verbatim}
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
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 - a
\end{verbatim}

`is_noetherian()`

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

**EXAMPLES:**
sage: Zp(5).is_noetherian()    #optional - sage.rings.padics
True

class sage.rings.ring.Ring
Bases: ParentWithGens

Generic ring class.

base_extend(R)

EXAMPLES:

sage: QQ.base_extend(GF(7))    #optional - sage.rings.finite_rings
Traceback (most recent call last):
... TypeError: no base extension defined
sage: ZZ.base_extend(GF(7))    #optional - sage.rings.finite_rings
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!    # optional - sage.combinat sage.modules
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
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

epsilon()  

Return the precision error of elements in this ring.

EXAMPLES:
For exact rings, zero is returned:

```
sage: ZZ.epsilon()
0
```

This also works over derived rings:

```
sage: RR['x'].epsilon()
2.22044604925031e-16
sage: QQ['x'].epsilon()
0
```

For the symbolic ring, there is no reasonable answer:

```
sage: SR.epsilon() #optional - sage.symbolics
Traceback (most recent call last):
  ... Not Implemented Error
```

### ideal(*args, **kwds)

Return the ideal defined by x, i.e., generated by x.

**INPUT:**

- *x* – list or tuple of generators (or several input arguments)
- coerce – bool (default: True); this must be a keyword argument. Only set it to False if you are certain that each generator is already in the ring.
- ideal_class – callable (default: 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 Ideal_generic or Ideal_nc.
- Further named arguments (such as side in the case of non-commutative rings) are forwarded to the ideal class.

**EXAMPLES:**

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

ideal_monoid()

Return the monoid of ideals of this ring.

EXAMPLES:

sage: F.<x,y,z> = FreeAlgebra(ZZ, 3)  # optional - sage.combinat sage.modules
sage: I = F * [x*y + y*z, x^2 + x*y - y*x - y^2] * F  # optional - sage.combinat sage.modules
sage: Q = F.quotient(I)  # optional - sage.combinat sage.modules
sage: Q.ideal_monoid()  # optional - sage.combinat sage.modules
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')  # optional - sage.combinat sage.modules
sage: I = F * [x*y + y*z, x^2 + x*y - y*x - y^2] * F  # optional - sage.combinat sage.modules
sage: Q = F.quo(I)  # optional - sage.combinat sage.modules
sage: Q.ideal_monoid()  # optional - sage.combinat sage.modules
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^2 + x*y - y*x - y^2)

is_commutative()

Return True if this ring is commutative.

EXAMPLES:

sage: QQ.is_commutative()  # False
sage: QQ['x,y,z'].is_commutative()  # True
sage: Q.<i,j,k> = QuaternionAlgebra(QQ, -1, -1)  # optional - sage.combinat sage.modules
sage: Q.is_commutative()  # False

is_exact()

Return True if elements of this ring are represented exactly, i.e., there is no precision loss when doing arithmetic.
**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()  # optional - 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:

```
sage: R.<a,b> = QQ[]
sage: S.<x,y> = R.quo((b^3))  # optional - sage.libs.singular
sage: S.is_field(proof=True)  # optional - sage.libs.singular
Traceback (most recent call last):
  ...
NotImplementedError
sage: S.is_field(proof=False)
```

(continues on next page)
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:

```python
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()  # optional - sage.rings.padics
True
sage: Qp(7).is_integral_domain()  # optional - sage.rings.padics
True
sage: R.<a,b> = ZZ[]
sage: S.<x,y> = R.quo((b^3))  # optional - sage.libs.singular
sage: S.is_integral_domain()  # optional - sage.libs.singular
False
```

This illustrates the use of the proof parameter:

```python
sage: R.<a,b> = ZZ[]
sage: S.<x,y> = R.quo((b^3))  # optional - sage.libs.singular
sage: S.is_integral_domain(proof=True)  # optional - sage.libs.singular
Traceback (most recent call last):
... NotImplementedError
sage: S.is_integral_domain(proof=False)  # optional - sage.libs.singular
False
```

is_noetherian()
Return True if this ring is Noetherian.
EXAMPLES:

```python
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:

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

```python
sage: ZZ.is_subring(QQ)
True
sage: ZZ.is_subring(GF(19))  # optional - sage.rings.finite_rings
False
```

**one()**

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

EXAMPLES:

```python
sage: ZZ.one()
1
sage: QQ.one()
1
sage: QQ['x'].one()
1
```

The result is cached:
sage: ZZ.one() is ZZ.one()
True

order()
The number of elements of self.

EXAMPLES:

sage: GF(19).order()  #optional - sage.rings.finite_rings
19
sage: QQ.order()
+Infinity

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

EXAMPLES:

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 [-bound,bound].

INPUT:

- bound – integer (default: 2)

ALGORITHM:
Uses Python’s randint.

unit_ideal()
Return the unit ideal of this ring.

EXAMPLES:

sage: Zp(7).unit_ideal()  #optional - 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).

EXAMPLES:

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

The result is cached:

1.1. Rings 23
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)  # optional - sage.rings.number_field
zeta6
sage: CyclotomicField(3).zeta(3)  # optional - sage.rings.number_field
zeta3
sage: CyclotomicField(3).zeta(3).multiplicative_order()  # optional - sage.rings.number_field
3
sage: a = GF(7).zeta(); a  # optional - sage.rings.finite_rings
3
sage: a.multiplicative_order()  # optional - sage.rings.finite_rings
6
sage: a = GF(49,'z').zeta(); a
```
(continues on next page)
sage: a.multiplicative_order()  #optional - sage.rings.finite_rings
48
sage: a = GF(49,'z').zeta(2); a  #optional - sage.rings.finite_rings
6
sage: a.multiplicative_order()  #optional - sage.rings.finite_rings
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()  #optional - 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()  #optional - sage.rings.number_field
38
sage: GF(19).zeta_order()  #optional - sage.rings.finite_rings
18
sage: GF(5^3,'a').zeta_order()  #optional - sage.rings.finite_rings
124
sage: Zp(7, prec=8).zeta_order()  #optional - 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)  #optional - sage.modules
sage: is_Ring(MS)  #optional - sage.modules
True

1.1. Rings
1.2 Abstract base classes for rings

```python
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)  # optional - sage.rings.number_field
    True
    sage: isinstance(AA, sage.rings.abc.AlgebraicField)  # optional - sage.rings.number_field
    False

    By design, there is a unique direct subclass:

    sage: sage.rings.abc.AlgebraicField.__subclasses__()  # optional - sage.rings.number_field
    [<class 'sage.rings.qqbar.AlgebraicField_common'>]
    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)  # optional - sage.rings.number_field
    True
    sage: isinstance(AA, sage.rings.abc.AlgebraicField_common)  # optional - sage.rings.number_field
    True

    By design, other than the abstract subclasses AlgebraicField and AlgebraicRealField, there is only one direct implementation subclass:

    sage: sage.rings.abc.AlgebraicField_common.__subclasses__()  # optional - sage.rings.number_field
    [<class 'sage.rings.abc.AlgebraicField_common'>,
     <class 'sage.rings.abc.AlgebraicRealField'>,
     <class 'sage.rings.qqbar.AlgebraicField_common'>]
    sage: len(sage.rings.abc.AlgebraicField_common.__subclasses__()) <= 3
    True
```
class sage.rings.abc.AlgebraicRealField

   Abstract base class for AlgebraicRealField.

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

   EXAMPLES:

   sage: import sage.rings.abc
   sage: isinstance(QQbar, sage.rings.abc.AlgebraicRealField)  # optional - sage.rings.number_field
   False
   sage: isinstance(AA, sage.rings.abc.AlgebraicRealField)     # optional - sage.rings.number_field
   True

   By design, there is a unique direct subclass:

   sage: sage.rings.abc.AlgebraicRealField.__subclasses__()     # optional - sage.rings.number_field
   [<class 'sage.rings.qqbar.AlgebraicRealField'>]
   sage: len(sage.rings.abc.AlgebraicRealField.__subclasses__()) <= 1  # optional - sage.rings.number_field
   True

class sage.rings.abc CallableSymbolicExpressionRing

   Abstract base class for CallableSymbolicExpressionRing_class.

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

   EXAMPLES:

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

   By design, there is a unique direct subclass:

   sage: sage.rings.abc.CallableSymbolicExpressionRing.__subclasses__()  # optional - sage.symbolic
   [<class 'sage.symbolic.callable.CallableSymbolicExpressionRing_class'>]
   sage: len(sage.rings.abc.CallableSymbolicExpressionRing.__subclasses__()) <= 1
   True

class sage.rings.abc.ComplexBallField

   Abstract base class for ComplexBallField.

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

   EXAMPLES:
sage: import sage.rings.abc
sage: isinstance(CBF, sage.rings.abc.ComplexBallField)
True

By design, there is a unique direct subclass:

sage: sage.rings.abc.ComplexBallField.__subclasses__()
[<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:

sage: import sage.rings.abc
sage: isinstance(CDF, sage.rings.abc.ComplexDoubleField)
True

By design, there is a unique direct subclass:

sage: sage.rings.abc.ComplexDoubleField.__subclasses__()
[<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)
True

By design, there is a unique direct subclass:

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

class sage.rings.abc.ComplexIntervalField
Bases: Field

28 Chapter 1. Base Classes for Rings, Algebras and Fields
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)
True
```

By design, there is a unique direct subclass:

```
sage: sage.rings.abc.ComplexIntervalField.__subclasses__() =[<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)  # optional - sage.rings.number_field
sage: isinstance(K, sage.rings.abc.NumberField_cyclotomic)  # optional - sage.rings.number_field
True
```

By design, there is a unique direct subclass:
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:

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

By design, there is a unique direct subclass:

```python
gsage: 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:

```python
gsage: import sage.rings.abc
gsage: issubclass(K, sage.rings.abc.NumberField_quadratic)
True
```

By design, there is a unique direct subclass:

```python
gsage: 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:

```python
gsage: import sage.rings.abc
gsage: K.<a> = NumberField(x^2 + 1); O = K.order(2*a)
gsage: issubclass(O, sage.rings.abc.Order)
True
```

By design, there is a unique direct subclass:

```python
gsage: len(sage.rings.abc.Order.__subclasses__()) <= 1
True
```
### 1.2. Abstract base classes for rings

**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:**

```python
sage: import sage.rings.abc
sage: isinstance(RBF, sage.rings.abc.RealBallField)
True
```

By design, there is a unique direct subclass:

```python
sage: sage.rings.abc.RealBallField.__subclasses__()
[<class 'sage.rings.real_arb.RealBallField'>]
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:**

```python
sage: import sage.rings.abc
sage: isinstance(RDF, sage.rings.abc.RealDoubleField)
True
```

By design, there is a unique direct subclass:

```python
sage: sage.rings.abc.RealDoubleField.__subclasses__()
[<class 'sage.rings.real_double.RealDoubleField_class'>]
sage: len(sage.rings.abc.RealDoubleField.__subclasses__()) <= 1
True
```

**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)
True
```
General Rings, Ideals, and Morphisms, Release 10.1

By design, there is a unique direct subclass:

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

By design, there is a unique direct subclass:

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

EXAMPLES:

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

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

```python
sage: sage.rings.abc.SymbolicRing.__subclasses__()
[<class 'sage.rings.abc.CallableSymbolicExpressionRing'>, ...
[<class 'sage.rings.abc.SymbolicRing'>]
sage: len(sage.rings.abc.SymbolicRing.__subclasses__()) <= 2
True
```

class `sage.rings.abc.UniversalCyclotomicField`

Bases: `Field`

Abstract base class for `UniversalCyclotomicField`. 

32 Chapter 1. Base Classes for Rings, Algebras and Fields
This class is defined for the purpose of \texttt{isinstance()} tests. It should not be instantiated.

**EXAMPLES:**

```
sage: import sage.rings.abc
sage: K = UniversalCyclotomicField()  # optional - sage.rings.number_field
  → optional - sage.rings.number_field
sage: isinstance(K, sage.rings.abc.UniversalCyclotomicField)  # optional - sage.rings.number_field
True
```

By design, there is a unique direct subclass:

```
sage: sage.rings.abc.UniversalCyclotomicField.__subclasses__()  # optional - sage.rings.number_field
  → optional - sage.rings.number_field
[<class 'sage.rings.universal_cyclotomic_field.UniversalCyclotomicField'>]
sage: len(sage.rings.abc.NumberField_cyclotomic.__subclasses__()) <= 1
True
```

**class** \texttt{sage.rings.abc.pAdicField}

Bases: \texttt{Field}

Abstract base class for \texttt{pAdicFieldGeneric}.

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

**EXAMPLES:**

```
sage: import sage.rings.abc
sage: isinstance(Zp(5), sage.rings.abc.pAdicField)  # optional - sage.rings.padics
False
sage: isinstance(Qp(5), sage.rings.abc.pAdicField)  # optional - sage.rings.padics
True
```

By design, there is a unique direct subclass:

```
sage: sage.rings.abc.pAdicField.__subclasses__()  # optional - sage.rings.padics
  → optional - sage.rings.padics
[<class 'sage.rings.padics.generic_nodes.pAdicFieldGeneric'>]
sage: len(sage.rings.abc.pAdicField.__subclasses__()) <= 1
True
```

**class** \texttt{sage.rings.abc.pAdicRing}

Bases: \texttt{EuclideanDomain}

Abstract base class for \texttt{pAdicRingGeneric}.

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

**EXAMPLES:**

```
sage: import sage.rings.abc
sage: isinstance(Zp(5), sage.rings.abc.pAdicRing)  # optional - sage.rings.padics
False
sage: isinstance(Qp(5), sage.rings.abc.pAdicRing)  # optional - sage.rings.padics
True
```
By design, there is a unique direct subclass:

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

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 \texttt{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.

\texttt{sage.rings.ideal.Cyclic}(\( R, n=None, \text{homog=False, singular=None} \))

Ideal of cyclic \( n \)-roots from 1-st \( n \) variables of \( R \) if \( R \) is coercible to \texttt{Singular}.

\textbf{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

\textbf{Note:} \( R \) will be set as the active ring in \texttt{Singular}

\textbf{EXAMPLES:}

An example from a multivariate polynomial ring over the rationals:

\begin{verbatim}
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()
[ x + y + z, y^2 + y*z + z^2, z^3 - 1]
\end{verbatim}

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

\begin{verbatim}
sage: R.<x,y,z,t,u,v> = QQ['x,y,z,t,u,v']
sage: I = sage.rings.ideal.Cyclic(R, 6)
\end{verbatim}

(continues on next page)
sage: B = I.groebner_basis()                      #optional - sage.libs.singular
sage: len(B)                                     #optional - sage.libs.singular
45

sage.rings.ideal.FieldIdeal(R)

Let $q = R.base_ring().order()$ and $(x_0, ..., x_n) = R.gens()$ 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)           #optional - sage.rings.finite_rings
sage: I = sage.rings.ideal.FieldIdeal(P); I         #optional - sage.rings.finite_rings
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)  #optional - sage.rings.finite_rings
sage: J = sage.rings.ideal.FieldIdeal(Q); J          #optional - sage.rings.finite_rings
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 \texttt{Ideal()} function:

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

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

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

EXAMPLES:
sage: R.<x> = ZZ[]
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: I
Ideal (x^2 + 3*x + 4, x^2 + 1) of Univariate Polynomial Ring in x over Integer Ring
sage: Ideal(R, [4 + 3*x + x^2, 1 + x^2])
Ideal (x^2 + 3*x + 4, x^2 + 1) of Univariate Polynomial Ring in x over Integer Ring
sage: Ideal((4 + 3*x + x^2, 1 + x^2))
Ideal (x^2 + 3*x + 4, x^2 + 1) of Univariate Polynomial Ring in x over Integer Ring
sage: ideal(x^2-2*x+1, x^2-1)
Ideal (x^2 - 2*x + 1, x^2 - 1) of Univariate Polynomial Ring in x over Integer Ring
sage: ideal([x^2-2*x+1, x^2-1])
Ideal (x^2 - 2*x + 1, x^2 - 1) of Univariate Polynomial Ring in x over Integer Ring
sage: l = [x^2-2*x+1, x^2-1]
sage: ideal(f^2 for f in l)
Ideal (x^4 - 4*x^3 + 6*x^2 - 4*x + 1, x^4 - 2*x^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).

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

sage: i
Ideal (1, t, t^2) of Univariate Polynomial Ring in t over Integer Ring
sage: 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:

sage: Ideal(3, 5)
Principal ideal (1) of Integer Ring
sage: Ideal(ZZ, 3, 5)
Principal ideal (1) of Integer Ring
sage: 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.

sage: I = Ideal([])
Traceback (most recent call last):
...
ValueError: unable to determine which ring to embed the ideal in

sage: I = 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.
```python
sage: I = R * (x^2 + 3)  
    #optional - sage.rings.finite_rings
sage: type(I)  
    #optional - sage.rings.finite_rings
<class 'sage.rings.polynomial.ideal.Ideal_1poly_field'>
```

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'>
```

### class `sage.rings.ideal.Ideal_fractional`

```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`

```python
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')[]  
    #optional - sage.rings.finite_rings
sage: I = R.ideal(t^4 + t + 1)  
    #optional - sage.rings.finite_rings
sage: I.absolute_norm()  
    #optional - 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:
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
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
sage: J.apply_morphism(psi)
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:
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()  # optional - 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()  # optional - 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]
```

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()                      #optional - sage.modules
S(0) ← S(-3) ← 0
```

2.1. Ideals of commutative rings

41
**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()  # optional - sage.libs.pari
True
sage: R.ideal(16).is_maximal()  # optional - sage.libs.pari
False
sage: S = Integers(8)
sage: S.ideal(0).is_maximal()  # optional - sage.libs.pari
False
sage: S.ideal(2).is_maximal()  # optional - sage.libs.pari
True
sage: S.ideal(4).is_maximal()  # optional - sage.libs.pari
False
```

**is_primary**

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()  # optional - sage.libs.singular
False
sage: J = I.primary_decomposition()[1]; J  # optional - sage.libs.singular
Ideal (y, x^2) of Multivariate Polynomial Ring in x, y over Rational Field
sage: J.is_primary()  # optional - sage.libs.singular
True
sage: J.is_prime()  # optional - sage.libs.singular
False
```

Some examples from the Macaulay2 documentation:
General Rings, Ideals, and Morphisms, Release 10.1

```
sage: R.<x, y, z> = GF(101)
    # optional - sage.rings.finite_rings
sage: I = R.ideal([y^6])
    # optional - sage.rings.finite_rings
sage: I.is_prime()  # optional - sage.libs.singular sage.rings.finite_rings
    True
sage: I.is_prime(R.ideal([y]))
    True
sage: I = R.ideal([x^4, y^7])
    # optional - sage.libs.singular sage.rings.finite_rings
sage: I.is_prime()  # optional - sage.libs.singular sage.rings.finite_rings
    True
sage: I = R.ideal([x*y, y^2])
    # optional - sage.libs.singular sage.rings.finite_rings
sage: I.is_prime()  # optional - sage.libs.singular sage.rings.finite_rings
    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])
    # a maximal ideal
sage: I.is_prime()  # a maximal ideal
    True
sage: I = R.ideal([x^2 - y])
    # a non-maximal prime ideal
sage: I.is_prime()  # a non-maximal prime ideal
    True
sage: I = R.ideal([x^2, y])
    # a non-prime primary ideal
sage: I.is_prime()  # a non-prime primary ideal
    False
sage: I = R.ideal([x^2, x*y])
    # a non-prime non-primary ideal
sage: I.is_prime()  # a non-prime non-primary ideal
    False
sage: S = Integers(8)
    # optional - sage.libs.singular
sage: S.ideal(0).is_prime()  # optional - sage.libs.singular
    False
sage: S.ideal(2).is_prime()  # optional - sage.libs.singular
    False
```

(continues on next page)
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):
  ...
NotImplementedError
```

**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:**

```
sage: R.<x> = ZZ[]
sage: I = R.ideal(2, x)
sage: I.is_principal()
Traceback (most recent call last):
  ...
NotImplementedError
```

**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:**

```
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:

```sage
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
sage: R.<t> = GF(8, names='a')[]  # optional - sage.rings.finite_rings
sage: I = R.ideal(t^4 + t + 1)  # optional - sage.rings.finite_rings
sage: I.norm()  # optional - 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
sage: R = ZZ['x']
sage: I = R.ideal(7)
sage: I.primary_decomposition()
Traceback (most recent call last):
... NotImplementedError
```

**random_element(**args, **kwds)**

Return a random element in this ideal.

EXAMPLES:

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

2.1. Ideals of commutative rings
**reduce(f)**

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

This is an element of \( R \) that is equivalent modulo \( I \) to \( f \) where \( I \) is \( \text{self} \).

**EXAMPLES:**

```sage
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
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
```

Note that \( \text{self.ring()} \) is different from \( \text{self.base_ring()} \)

```sage
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
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')
# ˓→optional - sage.libs.pari
sage: S = PolynomialRing(Rbar, 'y'); y = Rbar.gen(); S
# ˓→optional - 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)
# ˓→optional - sage.libs.pari
sage: J.ring()
# ˓→optional - sage.libs.pari
Univariate Quotient 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 \( \mathbb{Z} \):

\[
\begin{align*}
sage: & R = ZZ \\
sage: & I = R.ideal(42) \\
sage: & J = R.ideal(70) \\
sage: & I.gcd(J) \\
& \text{Principal ideal (14) of Integer Ring} \\
sage: & J.gcd(I) \\
& \text{Principal ideal (14) of Integer Ring}
\end{align*}
\]

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:

\[
\begin{align*}
sage: & R.<t> = GF(5)[] \\
& \text{#optional - sage.rings.finite_rings} \\
sage: & p = R.ideal(t^2 + 2) \\
& \text{#optional - sage.rings.finite_rings} \\
sage: & p.is_maximal() \\
& \text{#optional - sage.rings.finite_rings} \\
& \text{True} \\
sage: & p = R.ideal(t^2 + 1) \\
& \text{#optional - sage.rings.finite_rings} \\
sage: & p.is_maximal() \\
& \text{#optional - sage.rings.finite_rings} \\
& \text{False} \\
sage: & p = R.ideal(0) \\
& \text{#optional - sage.rings.finite_rings} \\
sage: & p.is_maximal() \\
& \text{#optional - sage.rings.finite_rings} \\
& \text{False}
\end{align*}
\]

(continues on next page)
sage: p = R.ideal(1)  # optional - sage.rings.finite_rings
sage: p.is_maximal()  # optional - sage.rings.finite_rings
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: ZZ.ideal(2).is_prime()  # optional - sage.libs.pari
True
sage: ZZ.ideal(-2).is_prime()  # optional - sage.libs.pari
True
sage: ZZ.ideal(4).is_prime()  # optional - sage.libs.pari
False
sage: ZZ.ideal(0).is_prime()  # optional - sage.libs.pari
True
sage: R.<x> = QQ[]
sage: P = R.ideal(x^2 + 1); P  # optional - sage.libs.pari
Principal ideal \((x^2 + 1)\) of Univariate Polynomial Ring in \(x\) over Rational Field
sage: P.is_prime()  # optional - sage.libs.pari
True

In fields, only the zero ideal is prime:

sage: RR.ideal(0).is_prime()  # optional - sage.libs.pari
True
sage: RR.ideal(7).is_prime()  # optional - sage.libs.pari
False

reduce(f)

Return the reduction of \(f\) modulo self.

EXAMPLES:

sage: I = 8*ZZ
sage: I.reduce(10)
2
sage: n = 10; n.mod(I)
2

residue_field()

Return the residue class field of this ideal, which must be prime.
Todo: Implement this for more general rings. Currently only defined for $\mathbb{Z}$ and for number field orders.

EXAMPLES:

```
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 valuation
sage: lift = F.lift_map(); lift
Lifting map:
  From: Residue field of Integers modulo 61
  To:   Integer Ring
sage: lift(F(12345/67890))
33
```

```
class sage.rings.ideal.Ideal_principal(ring, gens, coerce=True)
    Bases: Ideal_generic

A principal ideal.

See Ideal().

divides(other)
    Return True if self divides other.
```

```
EXAMPLES:

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
```

2.1. Ideals of commutative rings
**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:

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

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

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

```sage
R.<x> = ZZ[]
I = R.ideal(x)
J = R.ideal(2,x)
K = R.base_extend(QQ).ideal(2,x)
I
Principal ideal (x) of Univariate Polynomial Ring in x over Integer Ring
J
Ideal (2, x) of Univariate Polynomial Ring in x over Integer Ring
K
Principal ideal (1) of Univariate Polynomial Ring in x over Rational Field
I.is_principal()
True
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 \( R.n\text{gens}() \).
- \( \text{homog} \) – if True a homogeneous ideal is returned using the last variable in the ideal (default: False)
- \( \text{singular} \) – singular instance to use

**EXAMPLES:**

```plaintext
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
```

```plaintext
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:

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

Sage recognizes ideals of polynomial rings as well:

```plaintext
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
```

### 2.1. Ideals of commutative rings
2.2 Monoid of ideals in a commutative ring

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

```
sage: MS = MatrixSpace(QQ, 3, 3)  # optional - sage.modules
sage: type(MS.ideal(MS.one()).parent())  # optional - sage.modules
<class 'sage.rings.ideal_monoid.IdealMonoid_c_with_category'>
```

```
sage.rings.ideal_monoid.IdealMonoid(R)

Return the monoid of ideals in the ring R.

EXAMPLES:
```
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
```

```
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:
```
sage: R = QuadraticField(-23, 'a')  # optional - sage.rings.number_field
sage: from sage.rings.ideal_monoid import IdealMonoid
sage: M = IdealMonoid(R); M.ring()  # optional - sage.rings.number_field
True
```

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
```

(continues on next page)
General Rings, Ideals, and Morphisms, Release 10.1

(continued from previous page)

\[
\begin{bmatrix}
0 & 1 \\
-2 & 3
\end{bmatrix}
\]
of Full MatrixSpace of 2 by 2 dense matrices over Integer Ring

\[
sage: MS([0,1,-2,3])*MS
\]
Right Ideal

\[
\begin{bmatrix}
0 & 1 \\
-2 & 3
\end{bmatrix}
\]
of Full MatrixSpace of 2 by 2 dense matrices over Integer Ring

\[
sage: MS*MS([0,1,-2,3])*MS
\]
Twosided Ideal

\[
\begin{bmatrix}
0 & 1 \\
-2 & 3
\end{bmatrix}
\]
of Full MatrixSpace of 2 by 2 dense matrices over Integer Ring

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

\textbf{class} sage.rings.noncommutative_ideals.IdealMonoid_nc\( (R) \)

\textbf{Bases:} \texttt{IdealMonoid_c}

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

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

\textbf{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

\textbf{class} sage.rings.noncommutative_ideals.Ideal_nc\( (\text{ring}, \text{gens}, \text{coerce=True}, \text{side='twosided'}) \)

\textbf{Bases:} \texttt{Ideal_generic}

Generic non-commutative ideal.

All fancy stuff such as the computation of Groebner bases must be implemented in sub-classes. See \texttt{LetterplaceIdeal} for an example.

\textbf{EXAMPLES:}

\[
sage: MS = MatrixSpace(QQ,2,2)
sage: I = MS*[MS.1,MS.2]; I
\]
Left Ideal

\[
\begin{bmatrix}
0 & 1 \\
0 & 0
\end{bmatrix}
\]

(continues on next page)
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

\texttt{side()}

Return a string that describes the sidedness of this ideal.

\textbf{EXAMPLES:}

```
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 \( \mathbb{Z} \hookrightarrow \mathbb{Q} \):

```
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: 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: H = Hom(ZZ, GF(9, 'a'))
  # optional - sage.rings.finite_rings
sage: phi = H([1])
  # optional - sage.rings.finite_rings
sage: phi(5)
  # optional - sage.rings.finite_rings
2
sage: psi = H([4])
```

(continues on next page)
Map from single variable polynomial ring:

```python
sage: R.<x> = ZZ[]
sage: phi = R.hom([2], GF(5))
```

Identity map on the real numbers:

```python
sage: f = RR.hom([RR(1)]); f
```

Homomorphism from one precision of field to another.

From smaller to bigger doesn’t make sense:

```python
sage: R200 = RealField(200)
sage: f = RR.hom( R200 )
```

From bigger to small does:

```python
sage: f = RR.hom( RealField(15) )
```

Inclusion map from the reals to the complexes:
sage: i = RR.hom([CC(1)]); i
Ring morphism:
  From: Real Field with 53 bits of precision
  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: R.<x,y> = PolynomialRing(QQ)
sage: S.<a,b> = quo(R, ideal(1 + y^2))
sage: phi = S.hom([a^2, -b])
sage: 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()
sage: 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

(continues on next page)
Defn: Choice of lifting map

\[
\text{sage: }\pi.\text{lift}(13)
\]
\[
5
\]

Inclusion of $\text{GF}(2)$ into $\text{GF}(4, 'a')$:

\[
\text{sage: } k = \text{GF}(2) \quad #\text{optional - sage.rings.finite_rings}
\]
\[
\text{sage: } i = k.\text{hom}(\text{GF}(4, 'a')) \quad #\text{optional - sage.rings.finite_rings}
\]
\[
\text{sage: } i \quad #\text{optional - sage.rings.finite_rings}
\]

Ring morphism:
- From: Finite Field of size 2
- To: Finite Field in a of size $2^2$
  Defn: 1 |---> 1

\[
\text{sage: } i(0) \quad #\text{optional - sage.rings.finite_rings}
\]
\[
0
\]
\[
\text{sage: } a = i(1); a.\text{parent()} \quad #\text{optional - sage.rings.finite_rings}
\]

Finite Field in a of size $2^2$

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

\[
\text{sage: }\pi = \text{ZZ.}\text{hom}(k) \quad #\text{optional - sage.rings.finite_rings}
\]
\[
\text{sage: }\pi \quad #\text{optional - sage.rings.finite_rings}
\]

Natural morphism:
- From: Integer Ring
- To: Finite Field of size 2

\[
\text{sage: } f = i * \pi \quad #\text{optional - sage.rings.finite_rings}
\]
\[
\text{sage: } f \quad #\text{optional - sage.rings.finite_rings}
\]

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

\[
\text{sage: } a = f(5); a \quad #\text{optional - sage.rings.finite_rings}
\]
\[
1
\]
\[
\text{sage: } a.\text{parent()} \quad #\text{optional - sage.rings.finite_rings}
\]

Finite Field in a of size $2^2$
Inclusion from $\mathbb{Q}$ to the 3-adic field:

```sage
sage: phi = QQ.hom(Qp(3, print_mode='series'))
# optional - sage.rings.padics
sage: phi
# optional - sage.rings.padics
Ring morphism:
    From: Rational Field
    To:  3-adic Field with capped relative precision 20
sage: phi.codomain()
# optional - sage.rings.padics
3-adic Field with capped relative precision 20
sage: phi(394)
# optional - sage.rings.padics
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
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
sage: (1-sqrt2)^2
-2*sqrt2 + 3
sage: c = S.hom([1-sqrt2])  # this is not valid
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
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
sage: R.<t> = PowerSeriesRing(GF(5))
# optional - sage.rings.finite_rings
```

(continues on next page)
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)*f(b)
2 + 3*t^5 + 3*t^10 + t^15 + O(t^20)

Homomorphism of Laurent series ring:

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:
General Rings, Ideals, and Morphisms, Release 10.1

In SageMath, we can define a cyclotomic field and a homomorphism as follows:

```sage
K.<zeta7> = CyclotomicField(7)
c = K.hom([1/zeta7]); c
```

The output shows that `c` is a ring endomorphism of order 7 and degree 6, defined by:

```plaintext
zeta7 |--> -zeta7^5 - zeta7^4 - zeta7^3 - zeta7^2 - zeta7 - 1
```

We can also define an element `a` and its image under `c`:

```sage
a = (1+zeta7)^5; a
c(a)
```

Similarly, we can define another element `b = zeta7 + 1/zeta7` and its image under `c`:

```sage
c(zeta7 + 1/zeta7)
```

Embedding a number field into the reals:

```sage
R.<x> = PolynomialRing(QQ)
K.<beta> = NumberField(x^3 - 2)
alpha = RR(2)^(1/3); alpha
```

We can define a homomorphism `i` and its action on `beta`:

```sage
i = K.hom([alpha],check=False); i
```

An example from Jim Carlson:

```sage
K = QQ # by the way :-)
R.<a,b,c,d> = K []; R
S.<u> = K []; S
```

We can define a homomorphism `f` and its action on `a+b+c+d`:

```sage
f = R.hom([0,0,0,u], S); f
```

The class documentation for `FrobeniusEndomorphism_generic` is as follows:

```python
class sage.rings.morphism.FrobeniusEndomorphism_generic
    Bases: RingHomomorphism
```

3.1. Homomorphisms of rings
A class implementing Frobenius endomorphisms on rings of prime characteristic.

**power()**

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

**EXAMPLES:**

```python
sage: K.<u> = PowerSeriesRing(GF(5))
# optional - sage.rings.finite_rings
sage: Frob = K.frobenius_endomorphism()
# optional - sage.rings.finite_rings
sage: Frob.power()
# optional - sage.rings.finite_rings
1
sage: (Frob^9).power()
# optional - sage.rings.finite_rings
9
```

class sage.rings.morphism.RingHomomorphism

Bases: RingMap

Homomorphism of rings.

**inverse()**

Return the inverse of this ring homomorphism if it exists.

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:**

```python
sage: R.<t> = QQ[]
sage: f = R.hom([2*t - 1], R)
sage: f.inverse()
# optional - 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:

```python
sage: R.<x,y,z> = QQ[]
sage: f = R.hom([y*z, x*z, x*y], R)
sage: f.inverse()
# optional - sage.libs.singular
Traceback (most recent call last):
... Error: ring homomorphism not surjective
sage: f.is_injective()
# optional - sage.libs.singular
True
sage: Q.<x,y,z> = R.quotient(x*y*z - 1)
# optional - sage.libs.singular
```

(continues on next page)
sage: g = Q.hom([y*z, x*z, x*y], Q)  # _optional - sage.libs.singular
sage: g.inverse()  # _optional - sage.libs.singular
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: S.<x,y> = ZZ[]
sage: f = S.hom([x + 2*y, x + 3*y], S)
sage: f.inverse()  # _optional - 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()  # _optional - sage.libs.singular
True

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

sage: g = S.hom([x + y, x - y - 2], S)
sage: g.inverse()  # _optional - sage.libs.singular
Traceback (most recent call last):
  ... ZeroDivisionError: ring homomorphism not surjective
sage: R.<x,y> = QQ[x,y]
sage: h = R.hom([x + y, x - y - 2], R)
sage: (h.inverse() * h).is_identity()  # _optional - sage.libs.singular
True

This example by M. Nagata is a wild automorphism:

sage: R.<x,y,z> = QQ[]
sage: sigma = R.hom([x - 2*y*(z^2+x*y^2) - z*(z^2+x*y^2)^2, y + z*(z^2+x*y^2), z], R)
sage: tau = sigma.inverse(); tau  # _optional - sage.libs.singular
Ring endomorphism of Multivariate Polynomial Ring in x, y, z over Rational Field
Defn: x |--> -y^4*z + 2*x*y^2*z^2 - x^2*z^3 + 2*y^3 + 2*x*y*z + x
       y |--> -y^2*z - x*z^2 + y
       z |--> z
sage: (tau * sigma).is_identity()  # _optional - sage.libs.singular
True

3.1. Homomorphisms of rings
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.

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

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

```sage
K.<zeta7> = CyclotomicField(7)  #optional - sage.rings.number_field
c = K.hom([1/zeta7])  #optional - sage.rings.number_field
(c.inverse() * c).is_identity()  #optional - sage.rings.number_field
True
F.<t> = GF(7^3)  #optional - sage.rings.finite_rings
f = F.hom(t^7, F)  #optional - sage.rings.finite_rings
(f.inverse() * f).is_identity()  #optional - sage.rings.finite_rings
True
```

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

```sage
K.<i> = QuadraticField(-1)  #optional - sage.rings.number_field
A.<z,w> = K['z,w'].quotient('z*w - 1')  #optional - sage.rings.number_field
B.<x,y> = K['x,y'].quotient('x^2 + y^2 - 1')  #optional - sage.rings.number_field
```
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
sage: R.<x,y,z> = QQ[]
sage: S.<u,v> = QQ[]
sage: f = R.hom([u^2, u*v, v^2], S)
sage: I = S.ideal([u^6, u^5*v, u^4*v^2, u^3*v^3])
sage: J = f.inverse_image(I); J  #optional - sage.libs.singular
Ideal (y^2 - x*z, x*y*z, x^2*z, x^2*y, x^3)
of Multivariate Polynomial Ring in x, y, z over Rational Field

sage: f(J) == I  #optional - sage.libs.singular
True
```

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]]  #optional - sage.libs.singular
[x, x^2, x*y*z + y]
```

The image of the inverse image ideal can be strictly smaller than the original ideal:
Fractional ideals are not yet fully supported:

```
sage: K.<a> = NumberField(QQ['x']('x^2+2'))
                     #
                     ← optional - sage.rings.number_field
sage: f = K.hom([-a], K)
                     #
                     ← optional - sage.rings.number_field
sage: I = K.ideal([a + 1])
                     #
                     ← optional - sage.rings.number_field
sage: f.inverse_image(I)
                     #
                     ← optional - sage.rings.number_field
Traceback (most recent call last):
... NotImplementedError: inverse image not implemented...
```

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]

\texttt{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()
                     #
                     ← optional - sage.libs.singular
False
sage: Q.<x,y,z> = R.quotient(x*y*z - 1)
                     #
                     ← optional - sage.libs.singular
sage: Q.hom([y*z, x*z, x*y], Q).is_invertible()
                     #
                     ← optional - sage.libs.singular
```

(continues on next page)
ALGORITHM:
By default, this requires the computation of a Gröbner basis.

**is_surjective()**
Return whether this ring homomorphism is surjective.

**EXAMPLES:**

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

```python
sage: A.<x,y> = QQ[]
sage: B.<t> = QQ[]
sage: f = A.hom([t^4, t^3 - t^2], B)
sage: f.kernel()
optional - 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))
 optional - sage.libs.singular
 True
sage: Q = A.quotient(f.kernel())
 optional - sage.libs.singular
sage: Q.hom(f.im_gens(), B).is_injective()
 optional - sage.libs.singular
 True
```

The Steiner-Roman surface:

3.1. Homomorphisms of rings 67
sage: R.<x,y,z> = QQ[]
sage: S = R.quotient(x^2 + y^2 + z^2 - 1)  #
= optional - sage.libs.singular
sage: f = R.hom([x*y, x*z, y*z], S)  #
= optional - sage.libs.singular
sage: f.kernel()  #
= optional - 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:

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:

sage: R.<x,y> = QQ[]; S.<xx,yy> = R.quo([x^2, y^2]); f = S.cover()  #
= optional - sage.libs.singular
sage: f.pushforward(R.ideal([x, 3*x + x*y + y^2]))  #
= optional - sage.libs.singular
Ideal (xx, xx*yy + 3*xx) of Quotient of Multivariate Polynomial Ring
in x, y over Rational Field by the ideal (x^2, y^2)

class sage.rings.morphism.RingHomomorphism_coercion

Bases: RingHomomorphism

A ring homomorphism that is a coercion.

Warning: This class is obsolete. Set the category of your morphism to a subcategory of Rings instead.

class sage.rings.morphism.RingHomomorphism_cover

Bases: RingHomomorphism
A homomorphism induced by quotienting a ring out by an ideal.

EXAMPLES:

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

```plaintext
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:

```plaintext
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:

```plaintext
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:
```

(continues on next page)
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: p = (x - 4*y + 1/13)*t^2 + (1/2*x^2 - 1/3*y^2)*t + 2*y^2 + x
sage: Pf(p)
(-10*z + 1/13)*t^2 - z^2*t + 18*z^2 + 2*z

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

sage: MR = MatrixSpace(R, 2, 2)  # optional - sage.modules
sage: MS = MatrixSpace(S, 2, 2)  # optional - sage.modules
sage: M = MR([x^2 + 1/7*x*y - y^2, -1/2*y^2 + 2*y + 1/6,
           ....: 4*x^2 - 14*x, 1/2*y^2 + 13/4*x - 2/11*y])
sage: Mf = MR.hom(f, MS)  # optional - sage.modules
sage: Mf
Ring morphism:
From: Full MatrixSpace of 2 by 2 dense matrices over Multivariate Polynomial Ring in x, y over Rational Field
To: Full MatrixSpace of 2 by 2 dense matrices 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: Mf(M)
[ -29/7*z^2 -9/2*z^2 + 6*z + 1/6]
[ 16*z^2 - 28*z 9/2*z^2 + 131/22*z]

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

sage: MPR = MatrixSpace(PR, 2)  # optional - sage.modules
sage: MPS = MatrixSpace(PS, 2)  # optional - sage.modules
sage: M = MPR([(-x + y)*t^2 + 58*t - 3*x^2 + x*y,
            ....: (- 1/7*x*y - 1/40*x)*t^2 + (5*x^2 + y^2)*t + 2*y,
            ....: (- 1/3*y + 1)*t^2 + 1/3*x*y + y^2 + 5/2*y + 1/4,
            ....: (x + 6*y + 1)*t^2])
sage: MPf = MPR.hom(f, MPS); MPf
Ring morphism:
From: Full MatrixSpace of 2 by 2 dense matrices over Univariate Polynomial Ring in z over Rational Field
To: Full MatrixSpace of 2 by 2 dense matrices 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

(continues on next page)
Ring in \( t \) over Multivariate Polynomial Ring in \( x, y \) over Rational Field
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: \( x \mapsto 2z \)
\( y \mapsto 3z \)

\sage:`MPf(M)`

\texttt{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()`

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: \( a \mapsto 1/2x + 1/2y \)
\( b \mapsto 1/2x - 1/2y \)

\sage:`Pf.inverse()(Pf(x*t^2 + y*t))`

\texttt{underlying_map}()

Return the underlying homomorphism of the base ring.
EXAMPLES:

```python
sage: R.<x,y> = QQ[]
sage: S.<z> = QQ[]
sage: f = R.hom([2*z, 3*z], S)
sage: MR = MatrixSpace(R, 2)
    # optional - sage.modules
sage: MS = MatrixSpace(S, 2)
    # optional - sage.modules
sage: g = MR.hom(f, MS)
    # optional - sage.modules
sage: g.underlying_map() == f
    # optional - sage.modules
True
```

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()
    # optional - sage.libs.singular
sage: g.inverse()
    # optional - 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 \( R\.cover() \)

**OUTPUT:** a ring homomorphism

The domain \( R \) is a quotient object \( C \to R \), and \( R\.cover() \) is the ring homomorphism \( \varphi : C \to R \). The condition on the elements `im_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
class sage.rings.morphism.RingHomomorphism_from_quotient

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

(continues on next page)
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
   c |--> a
sage: phi(a + b + c)  # optional - sage.libs.singular

sage: loads(dumps(phi)) == phi  # optional - sage.libs.singular
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])  # optional - 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])  # optional - sage.libs.singular

sage: S.hom([yy,xx]).morphism_from_cover()  # optional - 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: R.<x> = ZZ[]
sage: K.<i> = NumberField(x^2 + 1)  # optional - sage.rings.number_field

sage: cc = K.hom([-i])  # optional - sage.rings.number_field

3.1. Homomorphisms of rings 73
optional - sage.rings.number_field

```
sage: S.<y> = K[]
optional - sage.rings.number_field
sage: phi = S.hom([y^2], base_map=cc)
optional - sage.rings.number_field
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
```
sage: phi(y)
optional - sage.rings.number_field
y^2
sage: phi(i*y)
optional - sage.rings.number_field
-i*y^2
```
```
sage: phi.base_map()
optional - sage.rings.number_field
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()
im_gens()
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[]
sage: f = R.hom([x, x + y])
sage: f.im_gens()
[x, x + y]
```

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

```
class sage.rings.morphism.RingMap
```

Bases: Morphism
Set-theoretic map between rings.

```python
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:

    ```python
sage: R.<x,y> = QQ[]
sage: S.<xbar,ybar> = R.quo( (x^2 + y^2, y) )  # optional - sage.libs.singular
    sage: S.lift()  # optional - 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, y)
    To:  Multivariate Polynomial Ring in x, y over Rational Field
    Defn: Choice of lifting map
    sage: S.lift() == 0  # optional - sage.libs.singular
    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:

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

`sage.rings.morphism.is_RingHomomorphism(phi)`

Return True if \( phi \) is of type `RingHomomorphism`.

EXAMPLES:

```python
sage: f = Zmod(8).cover()
sage: sage.rings.morphism.is_RingHomomorphism(f)
doctest:warning
...  DeprecationWarning: is_RingHomomorphism() should not be used anymore. Check whether your morphism is a subcategory of Rings() instead.
See https://github.com/sagemath/sage/issues/23204 for details.
True
```
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:

```
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:

```
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:

```sage
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
```

```python
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:

```sage
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
b
sage: phi(b)
    # needs sage.libs.singular
a
```

**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:

```sage
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
```

3.2. Space of homomorphisms between two rings
4.1 Quotient Rings

AUTHORS:

- William Stein
- Simon King (2011-04): Put it into the category framework, use the new coercion model.

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

\[
\text{sage: TestSuite(S).run(skip=['_test_nonzero_equal', '_test_elements', '_test_zero'])}
\]

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

\[
\text{sage: from sage.rings.noncommutative_ideals import Ideal_nc}
\text{sage: from itertools import product}
\text{sage: class PowerIdeal(Ideal_nc):
....: def __init__(self, R, n):
....: self._power = n
....: self._power = n
....: Ideal_nc.__init__(self, R, [R.prod(m) for m in product(R.gens(), repeat=n)])
....: def reduce(self,x):
....: R = self.ring()
....: return add([c*R(m) for m,c in x if len(m)<self._power],R(0))
\]

\[
\text{sage: F.<x,y,z> = FreeAlgebra(QQ, 3)}
\text{sage: I3 = PowerIdeal(F,3); I3}
\]

Two-sided Ideal \((x^3, x^2*y, x^2*z, x*y*x, x^y*z, x^z*x, x^z*y, x^z^2, y*x^2, y*x*y, y*x*z, y^2*x, y^3, y^2*z, y*z*x, y*z*y, y*z^2, z*x^2, z*x*y, z*x*z, z*y*x, z*y*z, z*y^2, z*y^3, z^2*x, z^2*y, z^3)\) 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:
General Rings, Ideals, and Morphisms, Release 10.1

```python
sage: Q3.<a,b,c> = F.quotient(I3)
#optional - sage.combinat sage.modules
sage: Q3
#optional - sage.combinat sage.modules
Quotient of Free Algebra on 3 generators (x, y, z) over Rational Field by
the ideal (x^3, x*y*x, x*y*z, x*y*z, x*z*x, x*z*y, x*z*z, y*x^2, y*x*y, y*x*z, y^2*x, y^3, y^2*z, y*z*x, y*z*y, y*z^2, z*x^2, z*x*y, z*y^2, z*z*x, z*z*y, z^3)

sage: (a+b+2)^4
#optional - sage.combinat sage.modules
16 + 32*a + 32*b + 24*a^2 + 24*a*b + 24*b*a + 24*b^2

sage: Q3.is_commutative()
#optional - sage.combinat sage.modules
False
```

Even though $Q_3$ is not commutative, there is commutativity for products of degree three:

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

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

```python
sage: I2 = PowerIdeal(F,2); I2
#optional - sage.combinat sage.modules
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)
#optional - sage.combinat sage.modules
sage: Q2.is_commutative()
#optional - sage.combinat sage.modules
True

sage: (a+b+2)^4
#optional - sage.combinat sage.modules
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:

```python
sage: from itertools import product
go
go
sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
#optional - sage.combinat sage.modules
sage: Q3 = F.quo(F*[F.prod(m) for m in product(F.gens(), repeat=3)]*F)
#optional - sage.combinat sage.modules
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
#optional - sage.combinat sage.modules
xbar*ybar - ybar*xbar
```

(continues on next page)
sage: Q3.0*(Q3.1*Q3.2) - (Q3.1*Q3.2)*Q3.0  
˓→ optional - sage.combinat sage.modules
sage: Q2 = F.quo(F.*[F.prod(m) for m in product(F.gens(), repeat=2)]*F)  
˓→ optional - sage.combinat sage.modules
sage: Q2.is_commutative()  
˓→ optional - sage.combinat sage.modules
True

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 names (if the quotient ring is a quotient of a polynomial ring). If 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: P.<x> = QQ[]
sage: R.<xx> = QuotientRing(P, P.ideal(x^2 + 1))  
˓→ optional - sage.libs.pari
sage: R  
˓→ optional - sage.libs.pari
```

(continues on next page)
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: P.<x> = QQ[]
sage: S = QuotientRing(P, P.ideal(x^2 - 2))  #→
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  #→
Univariate Quotient Polynomial Ring in xbar over Rational Field
with modulus x^2 + 1
sage: R.gens(); S.gens(); T.gens()  #→
(x, y)

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 $Q[x, y]$:

sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = QuotientRing(R, R.ideal(1 + y^2))  #→
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)
(0, d)
sage: for n in range(4): d^n
    → optional - sage.libs pari
1
d
-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, 0) 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:

sage: Zmod(9).ideal(-33)
Principal ideal (3) of Ring of integers modulo 9
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:

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:

sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
    → optional - sage.combinat sage.modules
sage: I = F * [x*y + y*z, x^2 + x*y - y*x - y^2] * F
    → optional - sage.combinat sage.modules
A quotient of a quotient is just the quotient of the original top ring by the sum of two ideals:

\[
\text{sage: } I = Q \star \{a^3 - b^3\} * Q
\]

\[
\text{sage: } R.<i,j,k> = Q.quo(I); R
\]

Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) over Rational Field by the ideal (-y*y*z - y*z*x - 2*y*z*z, x*y + y*z, x*x + x*y - y*x - y*y)

\[
\text{sage: } i^3
\]

\[
\text{sage: } j^3
\]

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

**EXAMPLES:**

\[
\text{sage: } R.<x> = PolynomialRing(ZZ, 'x')
\]

\[
\text{sage: } I = R.ideal([4 + 3*x + x^2, 1 + x^2])
\]

\[
\text{sage: } S = R.quotient_ring(I); S
\]

Quotient of Univariate Polynomial Ring in x over Integer Ring by the ideal (x^2 + 3*x + 4, x^2 + 1)

\[
\text{sage: } R.<x,y> = PolynomialRing(QQ)
\]

\[
\text{sage: } S.<a,b> = R.quo(x^2 + y^2)
\]

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

```python
sage: Q = QuotientRing(ZZ, 7 * ZZ)
sage: Q.ambient()
Integer Ring
```

```python
sage: P.<x> = QQ[]
sage: Q = QuotientRing(P, x^2 + 1)  # optional - sage.libs.pari
sage: Q.ambient()  # optional - 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):
  ... Not Implemented Error
```

### 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: sage: R.quotient_ring(I).construction()
```
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()
```

```python
sage: R.⟨x,y⟩ = PolynomialRing(QQ)
sage: Q = R.quo((x^2, y^2))
sage: pi = Q.cover()
sage: pi(x^3 + y)
#optional - sage.libs.singular
ybar
sage: l = pi.lift(x + y^3)
#optional - sage.libs.singular
```

cover_ring()

Returns the cover ring of the quotient ring: that is, the original ring $R$ from which we modded out an ideal,
EXAMPLES:

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

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

defining_ideal()

Returns the ideal generating this quotient ring.

EXAMPLES:

In the integers:

```python
sage: Q = QuotientRing(ZZ, 7 * ZZ)
sage: Q.defining_ideal()
Principal ideal (7) of Integer Ring
```

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

```python
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = QuotientRing(R, R.ideal(1 + y^2))  # optional - sage.libs.singular
sage: T.<c,d> = QuotientRing(S, S.ideal(a))  # optional - sage.libs.singular
sage: S.defining_ideal()  # optional - sage.libs.singular
Ideal (y^2 + 1) of Multivariate Polynomial Ring in x, y over Rational Field
sage: T.defining_ideal()  # optional - sage.libs.singular
Ideal (x, y^2 + 1) of Multivariate Polynomial Ring in x, y over Rational Field
```

gen(i=0)

Returns the i-th generator for this quotient ring.

EXAMPLES:

```python
sage: R = QuotientRing(ZZ, 7 * ZZ)
sage: R.gen(0)
1
```

```python
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = QuotientRing(R, R.ideal(1 + y^2))  # optional - sage.libs.singular
sage: T.<c,d> = QuotientRing(S, S.ideal(a))  # optional - sage.libs.singular
```

(continues on next page)
ideal(*gens, **kwds)

Return the ideal of self with the given generators.

EXAMPLES:

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

The non-commutative case is more interesting:
```sage
F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')  
I = F * [x*y + y*z, x^2 + x*y - y*x - y^2] * F
Q = F.quo(I)
Q.is_commutative()

sage: Q.1*Q.2 == Q.2*Q.1

J = F * [x*y - y*x, x*z - z*x, y*z - z*y, x^3 - y^3] * F
R = F.quo(J)
R.is_commutative()

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 NotImplemented error.
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:
```
General Rings, Ideals, and Morphisms, Release 10.1

sage: Q.is_integral_domain(proof=False)  # optional - sage.singular
False

is_noetherian()
Return True if this ring is Noetherian.

EXAMPLES:

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

sage: P.<x> = QQ[]
sage: R = QuotientRing(P, x^2 + 1)  # optional - sage.libs.pari
sage: R.is_noetherian()  # optional - sage.libs.pari
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)  # optional - sage.libs.pari
sage: S.is_noetherian()  # optional - sage.libs.pari
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()  # optional - 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

(continues on next page)
lifting_map()
    Return the lifting map to the cover.

    EXAMPLES:

```
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S = R.quotient(x^2 + y^2)
sage: pi = S.cover(); pi
    # optional - 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: Natural quotient map
sage: L = S.lifting_map(); L
    # optional - 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: L(S.0)
    # optional - sage.libs.singular
    x
sage: L(S.1)
    # optional - sage.libs.singular
    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
    # optional - sage.libs.singular
    -xbar*ybar^2 + 2*ybar^2
sage: L(z)
    # optional - sage.libs.singular
    -x*y^2 + 2*y^2
sage: L(z) == x^3 + 2*y^2
    # optional - sage.libs.singular
    False
```

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

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

(continues on next page)
Set-theoretic ring morphism:

From: Quotient of Full MatrixSpace of 2 by 2 dense matrices over Finite Field of size 5 by the ideal

\[
\begin{pmatrix}
0 & 1 \\
0 & 0
\end{pmatrix},
\begin{pmatrix}
0 & 0 \\
1 & 1
\end{pmatrix}
\]

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:

```python
sage: R = QuotientRing(ZZ, 7*ZZ)
sage: R.gens(); R.ngens()
(1,)
1
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = QuotientRing(R, R.ideal(1 + y^2))
#optional - sage.libs.singular
sage: T.<c,d> = QuotientRing(S, S.ideal(a))
#optional - sage.libs.singular
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()
#optional - sage.libs.singular
(x, y)
(a, b)
(0, d)
sage: R.ngens(); S.ngens(); T.ngens()
#optional - sage.libs.singular
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:

```python
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S = R.quotient(x^2 + y^2)
sage: S.retract((x+y)^2) # optional - sage.libs.singular
2*xbar*ybar
```

`term_order()`

Return the term order of this ring.

EXAMPLES:

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

`sage.rings.quotient_ring.is_QuotientRing(x)`

Tests whether or not x inherits from QuotientRing_nc.

EXAMPLES:

```python
sage: from sage.rings.quotient_ring import is_QuotientRing
sage: R.<x> = PolynomialRing(ZZ, 'x')
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: S = R.quotient_ring(I)
sage: is_QuotientRing(S)
True
sage: is_QuotientRing(R)
False
```
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.quo((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.quo(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()    # optional - sage.libs.singular
(xbar, ybar)
```

We name each of the generators.

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

(continues on next page)
sage: (a^3 + b^2).lift()  # optional - sage.libs.singular
-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.quo(1 - x^y); type(a)  # optional - sage.libs.singular
<class 'sage.rings.quotient_ring.QuotientRing_generic_with_category.element_'>
sage: a*b  # optional - sage.libs.singular
1
sage: S(2).is_unit()  # optional - sage.libs.singular
True
```

Check that github issue #29469 is fixed:

```
sage: a.is_unit()  # optional - sage.libs.singular
True
sage: (a+b).is_unit()  # optional - sage.libs.singular
False
```

**lc()**

Return the leading coefficient of this quotient ring element.

**EXAMPLES:**

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

**lift()**

If self is an element of $R/I$, then return self as an element of $R$.

**EXAMPLES:**

```
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)  # optional - sage.libs.singular
<class 'sage.rings.quotient_ring.QuotientRing_generic_with_category.element_'>
```

(continues on next page)
### \texttt{lm()}  
Return the leading monomial of this quotient ring element.

**EXAMPLES:**

```python  
sage: R.<x,y,z> = PolynomialRing(GF(7), 3, order='lex')  
optional - sage.rings.finite_rings  
sage: I = sage.rings.ideal.FieldIdeal(R)  
optional - sage.rings.finite_rings  
sage: Q = R.quo(I)  
optional - sage.rings.finite_rings  
sage: f = Q(z*y + 2*x)  
optional - sage.rings.finite_rings  
sage: f.lm()  
optional - sage.rings.finite_rings  
xbar  
```

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

**EXAMPLES:**

```python  
sage: R.<x,y,z> = PolynomialRing(GF(7), 3, order='lex')  
optional - sage.rings.finite_rings  
sage: I = sage.rings.ideal.FieldIdeal(R)  
optional - sage.rings.finite_rings  
sage: Q = R.quo(I)  
optional - sage.rings.finite_rings  
sage: f = Q(z*y + 2*x)  
optional - sage.rings.finite_rings  
sage: f.lt()  
optional - sage.rings.finite_rings  
2*xbar  
```

### \texttt{monomials()}  
Return the monomials in \texttt{self}.

**OUTPUT:**

A list of monomials.

**EXAMPLES:**

```python  
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)  
optional - sage.libs.singular  
<class 'sage.rings.quotient_ring.quotient_ring.QuotientRing_generic_with_category.element_'  
(continues on next page)  
```
reduce\((G)\)
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:

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

Notice that the result above is not minimal:

```sage
sage: I2.reduce(f)  # optional - sage.rings.finite_rings
```

variables()
Return all variables occurring in \(self\).

OUTPUT:
A tuple of linear monomials, one for each variable occurring in \(self\).

EXAMPLES:
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)  
    # optional - sage.libs.singular  
<class 'sage.rings.quotient_ring.QuotientRing_generic_with_category.element_class'>

sage: a.variables()  
    # optional - sage.libs.singular  
(a,)

sage: b.variables()  
    # optional - sage.libs.singular  
(b,)

sage: s = a^2 + b^2 + 1; s  
    # optional - sage.libs.singular  
1

sage: s.variables()  
    # optional - sage.libs.singular  
()

sage: (a + b).variables()  
    # optional - sage.libs.singular  
(a, b)
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: 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:

```
sage: FractionField(IntegerRing())
Rational Field
sage: FractionField(PolynomialRing(RationalField(),'x'))
```

(continues on next page)
Fraction Field of Univariate Polynomial Ring in x over Rational Field
\texttt{sage: FractionField(PolynomialRing(IntegerRing(),'x'))}
Fraction Field of Univariate Polynomial Ring in x over Integer Ring
\texttt{sage: FractionField(PolynomialRing(RationalField(),2,'x'))}
Fraction Field of Multivariate Polynomial Ring in x0, x1 over Rational Field

Dividing elements often implicitly creates elements of the fraction field:

\begin{verbatim}
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
\end{verbatim}

The input must be an integral domain:

\begin{verbatim}
sage: Frac(Integers(4))
Traceback (most recent call last):
 ...
TypeError: R must be an integral domain.
\end{verbatim}

\texttt{class sage.rings.fraction_field.FractionFieldEmbedding}

\texttt{Bases: DefaultConvertMap_unique}

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

\texttt{EXAMPLES:}

\begin{verbatim}
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
\end{verbatim}

\texttt{is_injective()}

Return whether this map is injective.

\texttt{EXAMPLES:}

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

\begin{verbatim}
sage: R.<x> = QQ[]
sage: R.fraction_field().coerce_map_from(R).is_injective()
True
\end{verbatim}

\texttt{is_surjective()}

Return whether this map is surjective.

\texttt{EXAMPLES:}

\begin{verbatim}
sage: R.<x> = QQ[]
sage: R.fraction_field().coerce_map_from(R).is_surjective()
False
\end{verbatim}
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(R, element_class=<class 'sage.rings.fraction_field_element.FractionFieldElement_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: R.fraction_field() # optional - sage.rings.finite_rings
```
```
sage: K = R.fraction_field() # optional - sage.rings.finite_rings
```
```
sage: K.function_field() # optional - sage.rings.finite_rings
```
Rational function field in t over Finite Field of size 5

5.1. Fraction Field of Integral Domains
maximal_order()

Return the maximal order in this fraction field.

EXAMPLES:

```python
sage: K = FractionField(GF(5)['t'])
# optional - sage.rings.finite_rings
sage: K.maximal_order()  # optional - sage.rings.finite_rings
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'])
# optional - sage.rings.finite_rings
sage: K.ring_of_integers()  # optional - sage.rings.finite_rings
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'])
```

characteristic()

Return the characteristic of this fraction field.

EXAMPLES:

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

sage: R = Frac(ZZ['t']); R.characteristic()
0
sage: R = Frac(GF(5)['w']); R.characteristic()  # optional - sage.rings.finite_rings
5
construction()

EXAMPLES:

```python
sage: Frac(ZZ['x']).construction()
(FractionField, Univariate Polynomial Ring in x over Integer Ring)
sage: K = Frac(GF(3)['t'])
# optional - sage.rings.finite_rings
sage: f, R = K.construction()
# optional - sage.rings.finite_rings
sage: f(R)
Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 3
sage: f(R) == K
# optional - sage.rings.finite_rings
True
```

```python
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()
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:

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

`ngens()`

This is the same as for the parent object.

EXAMPLES:

```sage
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
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
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
sage: R.<x> = QQ[]
sage: R.fraction_field().some_elements()
[0, 1,
```

(continues on next page)
\[
\begin{align*}
&x, \\
&2x, \\
&x/(x^2 + 2x + 1), \\
&1/x^2, \\
&... \\
&(2x^2 + 2)/(x^2 + 2x + 1), \\
&(2x^2 + 2)/x^3, \\
&(2x^2 + 2)/(x^2 - 1), \\
&2 \\
\end{align*}
\]

sage.rings.fraction_field.is_FractionField(x)

Test whether or not \(x\) inherits from \texttt{FractionField\_generic}.

EXAMPLES:

\begin{verbatim}
    sage: from sage.rings.fraction_field import is_FractionField
    sage: is_FractionField(Frac(ZZ['x']))
    True
    sage: is_FractionField(QQ)
    False
\end{verbatim}

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:

\begin{verbatim}
    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
\end{verbatim}

denominator()

Return the denominator of self.

EXAMPLES:
General Rings, Ideals, and Morphisms, Release 10.1

```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()
sage: (x/x).is_one()
True
sage: (x/y).is_one()
False
```

**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:**

```python
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[]
```

(continues on next page)
General Rings, Ideals, and Morphisms, Release 10.1

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

is_zero()

Return True if this element is equal to zero.

EXAMPLES:

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.

5.2. Fraction Field Elements 107
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:**

```python
sage: R.<x> = RealField(10)[]
sage: f = (x^2+2*x+1)/(x+1); f
(x^2 + 2.0*x + 1.0)/(x + 1.0)
sage: f.reduce(); f
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:**

```python
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:**

```python
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() #optional - sage.libs.pari
[t - 1, t + 3, t^2 + 2, t^2 + t + 1, t^4 - 2]
```

```

**sage.rings.fraction_field_element.is_FractionFieldElement(x)**

Return whether or not \( x \) is a \textit{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
```

**sage.rings.fraction_field_element.make_element(parent, numerator, denominator)**

Used for unpickling \textit{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)
```

**sage.rings.fraction_field_element.make_element_old(parent, cdict)**

Used for unpickling old \textit{FractionFieldElement} pickles.
EXAMPLES:

```python
sage: from sage.rings.fraction_field_element import make_element_old
sage: R.<x,y> = ZZ[]
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: LZ = Localization(ZZ, (5,11))
```

Full MatrixSpace of 2 by 2 dense matrices over Integer Ring localized at (5, 11)

```sage
sage: m = matrix(LZ, [[5, 7], [0,11]])
```

Full MatrixSpace of 2 by 2 dense matrices over Integer Ring localized at (5, 11)

```sage
sage: m.parent()  # parent of inverse is different: see documentation of m.__invert__
```

Optional - sage.modules

```sage
sage: m.parent()  # optional - sage.modules
```

Optional - sage.modules

```sage
sage: ~m
```

# parent of inverse is different: see documentation of m.__invert__

Optional - sage.modules

```sage
sage: ~m
```

Optional - sage.modules

```sage
sage: ~m
```

Optional - sage.modules

```sage
sage: ~m
```

Optional - sage.modules

```sage
sage: ~m
```

Optional - sage.modules

```sage
sage: ~m
```

Optional - sage.modules

```sage
sage: mi = matrix(LZ, ~m)
```

Optional - sage.modules

```sage
sage: mi.parent()  # optional - sage.modules
```

Optional - sage.modules

```sage
sage: mi == ~m
```

Optional - sage.modules

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[
```

Optional - sage.modules

```sage
sage: u = [u0, u1, u2]
```

Optional - sage.modules

```sage
sage: S = Set(u)
```

Optional - sage.modules

```sage
sage: I = S.cartesian_product(S)
```

Optional - sage.modules

```sage
sage: add_units = u + [q, q + 1] + [ui - uj for ui, uj in I if ui != uj]
```

Optional - sage.modules

```sage
sage: add_units += [q ui - uj for ui, uj in I if ui != uj]
```

Optional - sage.modules

```sage
sage: L = R.localization(tuple(add_units)); L
```

(continues on next page)
Define the representation matrices (of one of the three dimensional irreducible representations):

```
sage: m1 = matrix(L, [[u1, 0, 0], [0, u0, 0], [0, 0, u0]])
#optional - sage.modules
sage: m2 = matrix(L, [[(u0*q - u0)/(u0 - u1), (u0*q - u1)/(u0 - u1), 0], ...
....: [(-u1*q + u0)/(u0 - u1), (-u1*q + u1)/(u0 - u1), 0], ...
....: [0, 0, -1]])
#optional - sage.modules
sage: m3 = matrix(L, [[-1, 0, 0], ...
....: [0, u0*(1 - q)/(u1*q - u0), q*(u1 - u0)/(u1*q - u0)], ...
....: [0, (u1*q^2 - u0)/(u1*q - u0), (u1*q^2 - u1*q)/(u1*q - u0)]])
#optional - sage.modules
sage: m1.base_ring() == L
#optional - sage.modules
True
```

Check relations of the Ariki-Koike algebra:

```
sage: m1*m2*m1*m2 == m2*m1*m2*m1
#optional - sage.modules
True
sage: m2*m3*m2 == m3*m2*m3
#optional - sage.modules
True
sage: m1*m3 == m3*m1
#optional - sage.modules
True
sage: m1**3 - (u0+u1+u2)*m1**2 + (u0*u1+u0*u2+u1*u2)*m1 - u0*u1*u2 == 0
#optional - sage.modules
True
sage: m2**2 - (q-1)*m2 - q == 0
#optional - sage.modules
True
sage: m3**2 - (q-1)*m3 - q == 0
#optional - sage.modules
True
sage: ~m1 in m1.parent()
#optional - sage.modules
True
sage: ~m2 in m2.parent()
#optional - sage.modules
True
sage: ~m3 in m3.parent()
#optional - sage.modules
True
```

Obtain specializations in positive characteristic:
sage: Fp = GF(17)  # optional - sage.rings.finite_rings

sage: f = L.hom((3,5,7,11), codomain=Fp); f  # optional - sage.rings.finite_rings
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:   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  # optional - sage.modules sage.rings.finite_rings
[5 0 0]
[0 3 0]
[0 0 3]

sage: mFp1.base_ring()  # optional - sage.modules sage.rings.finite_rings
Finite Field of size 17

sage: mFp2 = matrix({k: f(v) for k, v in m2.dict().items()}); mFp2  # optional - sage.modules sage.rings.finite_rings
[ 2 3 0]
[ 9 8 0]
[ 0 0 16]

sage: mFp3 = matrix({k: f(v) for k, v in m3.dict().items()}); mFp3  # optional - sage.modules sage.rings.finite_rings
[16 0 0]
[ 0 4 5]
[ 0 7 6]

Obtain specializations in characteristic 0:

sage: fQ = L.hom((3,5,7,11), codomain=QQ); fQ  # optional - sage.rings.finite_rings
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: mQ1 = matrix({k: fQ(v) for k, v in m1.dict().items()}); mQ1  # optional - sage.modules sage.rings.finite_rings
[5 0 0]
[0 3 0]
[0 0 3]

sage: mQ1.base_ring()  # optional - sage.modules sage.rings.finite_rings
Rational Field

(continues on next page)
```
sage: mQ2 = matrix({k: fQ(v) for k, v in m2.dict().items()}); mQ2
[[-15 -14  0]
 [ 26  25  0]
 [  0  0 -1]]
sage: mQ3 = matrix({k: fQ(v) for k, v in m3.dict().items()}); mQ3
[[-1  0  0]
 [ 0 -15/26 11/26]
 [ 0 301/26 275/26]]
sage: S.<x, y, z, t> = QQ[]
sage: T = S.quo(x + y + z)
sage: F = T.fraction_field(); F
Fraction Field of Quotient of Multivariate Polynomial Ring in x, y, z, t
over Rational Field by the ideal (x + y + z)
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, 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
    u2 |--> zbar
    q |--> tbar
sage: mF1 = matrix({k: fF(v) for k, v in m1.dict().items()}); mF1
[[ ybar  0  0]
 [ 0 -ybar - zbar  0]
 [ 0  0 -ybar - zbar]]
sage: mF1.base_ring() == F
True
```

AUTHORS:

- Sebastian Oehms 2019-12-09: initial version.
- Sebastian Oehms 2022-03-05: fix some corner cases and add factor() (github issue #33463)

```python
class sage.rings.localization.Localization(base_ring, extra_units, names=None, normalize=True, category=None, warning=True)
```

Bases: IntegralDomain, UniqueRepresentation

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 \( R \rightarrow R[S^{-1}] \) 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
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)) # optional - sage.rings.padic
Traceback (most recent call last):
... ValueError: all given elements are invertible in 7-adic Ring with capped relative precision 20

sage: R.<x> = ZZ[]
sage: L = R.localization(x**2 + 1) # optional - sage.libs.pari
sage: s = (x+5)/(x**2+1)
sage: s in L # optional - sage.libs.pari
```

(continues on next page)
True

```python
sage: t = (x+5)/(x**2+2)
sage: t in L
    →optional - sage.libs.pari
False
sage: L(t)
    →optional - sage.libs.pari
Traceback (most recent call last):
... TypeError: fraction must have unit denominator
```

```python
sage: L(s) in R
    →optional - sage.libs.pari
False
sage: y = L(x)
    →optional - sage.libs.pari
sage: g = L(s)
    →optional - sage.libs.pari
sage: g.parent()
    →optional - sage.libs.pari
Univariate Polynomial Ring in x over Integer Ring localized at (x^2 + 1,)
```  

```python
sage: f = (y+5)/(y**2+1); f
    →optional - sage.libs.pari
(x + 5)/(x^2 + 1)
sage: f == g
    →optional - sage.libs.pari
True
sage: (y+5)/(y**2+2)
    →optional - sage.libs.pari
Traceback (most recent call last):
... ValueError: factor x^2 + 2 of denominator is not a unit
```

```python
sage: Lau.<u, v> = LaurentPolynomialRing(ZZ)
sage: LauL = Lau.localization(u + 1)
sage: LauL(~u).parent()
Multivariate Polynomial Ring in u, v over Integer Ring localized at (v, u, u + 1)
```

More examples will be shown typing `sage.rings.localization`?

**Element**

alias of `LocalizationElement`

**characteristic()**

Return the characteristic of `self`.

**EXAMPLES:**

```python
sage: R.<a> = GF(5)[]
    →optional - sage.rings.finite_rings
sage: L = R.localization((a**2 - 3, a))
    →optional - sage.rings.finite_rings
sage: L.characteristic()
    →optional - sage.rings.finite_rings
5
```
fraction_field()
Return the fraction field of self.

EXAMPLES:

```sage
sage: R.<a> = GF(5)[]  #
              ^optional - sage.rings.finite_rings
sage: L = Localization(R, (a**2 - 3, a))  #
              ^optional - sage.rings.finite_rings
sage: L.fraction_field()  #
              ^optional - sage.rings.finite_rings
Fraction Field of Univariate Polynomial Ring in a over Finite Field of size 5
sage: L.is_subring(_)
True
```

```sage
sage: R.<x, y> = ZZ[]
sage: R.localization((x**2 + 1, y - 1)).gen(0)
              ^optional - sage.libs.pari
x
sage: ZZ.localization(2).gen(0)
1
```

gen(i)
Return the i-th generator of self which is the i-th generator of the base ring.

EXAMPLES:

```sage
sage: R.<x, y> = ZZ[]
sage: R.localization((x**2 + 1, y - 1)).gen(0)
              ^optional - sage.libs.pari
x
sage: ZZ.localization(2).gen(0)
1
```

gens()
Return a tuple whose entries are the generators for this object, in order.

EXAMPLES:

```sage
sage: R.<x, y> = ZZ[]
sage: Localization(R, (x**2 + 1, y - 1)).gens()
              ^optional - 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:
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.is_field()
False
sage: R.krull_dimension()
1
```

gens()
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)).gens()
2
sage: Localization(ZZ, 2).gens()
1
```

class sage.rings.localization.LocalizationElement(parent, x)
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:

```python
sage: from sage.rings.localization import LocalizationElement
sage: P.<x,y,z> = GF(5)[]  
# optional - sage.rings.finite_rings
sage: L = P.localization((x, y*z-x))  
# optional - sage.rings.finite_rings
sage: LocalizationElement(L, 4/(y*z-x)**2)  
# optional - sage.rings.finite_rings
(-1)/(y^2*z^2 - 2*x*y*z + x^2)
sage: _.parent()  
# optional - sage.rings.finite_rings
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['x, y']
sage: L = P.localization(X - Y)
sage: x, y = L.gens()
sage: p = (x^2 - y^2)/(x-y)^2  # optional - sage.libs.singular
sage: p.factor()  # optional - 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()  # optional - sage.libs.singular
1/(x*y*z)
sage: L(z).inverse_of_unit()  # optional - sage.libs.singular
1/z
```

is_unit()  
Return True if self is a unit.

EXAMPLES:

```
sage: P.<x,y,z> = QQ[]
sage: L = P.localization((x, y*z))  # optional - sage.libs.pari
sage: L(y*z).is_unit()  # optional - sage.libs.pari
True
sage: L(z).is_unit()  # optional - sage.libs.pari
True
sage: L(x*y*z).is_unit()  # optional - sage.libs.pari
True
```

(continues on next page)
\textbf{numerator()}

Return the numerator of self.

\textbf{EXAMPLES:}

\begin{verbatim}
  sage: L = ZZ.localization((3,5))
  sage: L(7/15).numerator()
  7
\end{verbatim}

\textbf{sage.rings.localization.normalize_extra_units}(\textit{base_ring}, \textit{add_units}, \textit{warning}=\textit{True})

Function to normalize input data. The given list will be replaced by a list of the involved prime factors (if possible).

\textbf{INPUT:}

\begin{itemize}
  \item \textit{base_ring} – an instance of \texttt{IntegralDomain}
  \item \textit{add_units} – list of elements from base ring
  \item \textit{warning} – (optional, default: \textit{True}) to suppress a warning which is thrown if no normalization was possible
\end{itemize}

\textbf{OUTPUT:}

List of all prime factors of the elements of the given list.

\textbf{EXAMPLES:}

\begin{verbatim}
  sage: from sage.rings.localization import normalize_extra_units
  sage: normalize_extra_units(ZZ, [3, -15, 45, 9, 2, 50])
  [2, 3, 5]
  sage: normalize_extra_units(P, [3*x, z*y**2, 2*z, 18*(x*y*z)**2, x*z, 6*x*z, 5])
  [2, 3, 5, z, y, x]
  sage: normalize_extra_units(Q, [3*x, z*y**2, 2*z, 18*(x*y*z)**2, x*z, 6*x*z, 5])
  [z, y, x]
\end{verbatim}
optional - sage.libs.pari

\[ b^2 - 5 \]
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 $F_{5^4}/F_{5^2}$:

```sage
GF(5^4).over(GF(5^2))
```

Field in $z_4$ with defining polynomial $x^2 + (4*z_2 + 3)*x + z_2$ over its base

By default, Sage reuses the canonical generator of the top ring (here $z_4 \in F_{5^4}$), together with its name. However, the user can customize them by passing in appropriate arguments:

```sage
F = GF(5^2)
k = GF(5^4)
z4 = k.gen()
K.<a> = k.over(F, gen = 1-z4)
K
```

Field in $a$ with defining polynomial $x^2 + z_2*x + 4$ over its base

The base of the extension is available via the method `base()` (or equivalently `base_ring()`):

```sage
K.base()
```

Finite Field in $z_2$ of size $5^2$

It is also possible to build an extension on top of another extension, obtaining this way a tower of extensions:

```sage
L.<b> = GF(5^8).over(K)
L
```

Field in $b$ with defining polynomial $x^2 + (4*z_2 + 3*a)*x + 1 - a$ over its base

```sage
L.base()
```

Field in $a$ with defining polynomial $x^2 + z_2*x + 4$ over its base

```sage
L.base().base()
```

Finite Field in $z_2$ of size $5^2$

The method `bases()` gives access to the complete list of rings in a tower:

```sage
L.bases()
```

[Field in $b$ with defining polynomial $x^2 + (4*z_2 + 3*a)*x + 1 - a$ over its base, (continues on next page)]
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:

```sage
L.basis_over(K)
[1, b]
L.basis_over(F)
[1, a, b, a*b]
```

When the base is omitted, the default is the natural base of the extension:

```sage
L.basis_over()
[1, b]
```

The method `sage.rings.ring_extension_element.RingExtensionWithBasis.vector()` computes the coordinates of an element according to the above basis:

```sage
u = a + 2*b + 3*a*b
t = u.vector()    # over K
(a, 2 + 3*a)
t = u.vector(F)   # over F
(0, 1, 2, 3)
```

One can also compute traces and norms with respect to any base of the tower:

```sage
u.trace()        # over K
(2*z2 + 1) + (2*z2 + 1)*a
u.trace(F)       # over F
z2 + 1
u.trace().trace() # over K, then over F
z2 + 1
u.norm()         # over K
(z2 + 1) + (4*z2 + 2)*a
u.norm(F)        # over F
2*z2 + 2
```

And minimal polynomials:

```sage
u.minpoly()      # over K
x^2 + ((3*z2 + 4) + (3*z2 + 4)*a)*x + (z2 + 1) + (4*z2 + 2)*a
u.minpoly(F)     # over F
x^4 + (4*z2 + 4)*x^3 + x^2 + (z2 + 1)*x + 2*z2 + 2
```

**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: x = polygen(ZZ, 'x')
sage: A.<a> = ZZ.extension(x^2 - 2)
sage: OK = A.over()
sage: K = OK.fraction_field()
sage: 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
```

```
sage: K.ring() is OK
True
```

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:

```python
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:

```python
sage: L.basis_over()
[1, c, c^2]
sage: K.basis_over()
[1, b]
```

Note that base must be an explicit base over which the extension has been defined (as listed by the method bases()):

```python
sage: L.degree_over(GF(5^6))
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: 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()
sage: K1
Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5
```

(continues on next page)
→ over its base
\texttt{sage: K1.bases()}
\[
\text{[Fraction Field of Order in Number Field in a with defining polynomial } x^2 - 5 \text{ over its base, Order in Number Field in a with defining polynomial } x^2 - 5 \text{ over its base, Integer Ring]}
\]

\texttt{sage: K2 = OK.fraction_field(extend_base=True)}
\texttt{sage: K2}
\text{Fraction Field of Order in Number Field in a with defining polynomial } x^2 - 5 \text{ over its base}
\texttt{sage: K2.bases()}
\[
\text{[Fraction Field of Order in Number Field in a with defining polynomial } x^2 - 5 \text{ over its base, Rational Field]}
\]

Note that there is no coercion map between $K_1$ and $K_2$:

\texttt{sage: K1.has_coerce_map_from(K2)}
\text{False}
\texttt{sage: K2.has_coerce_map_from(K1)}
\text{False}

We check that when the extension is a field, its fraction field does not change:

\texttt{sage: K1.fraction_field() is K1}
\text{True}
\texttt{sage: K2.fraction_field() is K2}
\text{True}

\texttt{free_module(base=None, map=True)}

Return a free module $V$ over $\text{base}$ which is isomorphic to this ring

\textbf{INPUT:}

- $\text{base}$ – a commutative ring (which might be itself an extension) or None (default: None)
- $\text{map}$ – boolean (default True); whether to return isomorphisms between this ring and $V$

\textbf{OUTPUT:}

- A finite-rank free module $V$ over $\text{base}$
- The isomorphism from $V$ to this ring corresponding to the basis output by the method $\text{basis_over()}$ (only included if \text{map} is True)
- The reverse isomorphism of the isomorphism above (only included if \text{map} is True)

\textbf{EXAMPLES:}

\texttt{sage: F = GF(11)}
\texttt{sage: K.<a> = GF(11^2).over()}  
\texttt{sage: L.<b> = GF(11^6).over(K)}

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 $(1, b, b^2)$:
Vector space of dimension 3 over Field in a with defining polynomial $x^2 + 7x + 2$ over its base

Generic map:
From: Vector space of dimension 3 over Field in a with defining polynomial $x^2 + 7x + 2$ over its base
To: Field in b with defining polynomial $x^3 + (7 + 2a)x^2 + (2 - a)x - a$ over its base

Generic map:
From: Field in b with defining polynomial $x^3 + (7 + 2a)x^2 + (2 - a)x - a$ over its base
To: Vector space of dimension 3 over Field in a with defining polynomial $x^2 + 7x + 2$ over its base

Similarly, one can view $L$ as a $F$-vector space of dimension 6:

In this case, the isomorphisms between $V$ and $L$ are given by the basis $(1, a, b, ab, b^2, ab^2)$:

When base is omitted, the default is the base of this extension:

Note that base must be an explicit base over which the extension has been defined (as listed by the method bases()):

A class for finite free ring extensions generated by a single element

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 case 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: 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()
sage: 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]
```

```python
sage: K2 = OK.fraction_field(extend_base=True)
sage: 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)
False
sage: K2.has_coerce_map_from(K1)
False
```

We check that when the extension is a field, its fraction field does not change:

```python
sage: K1.fraction_field() is K1
True
sage: K2.fraction_field() is K2
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)

**EXAMPLES:**

```python
```
General Rings, Ideals, and Morphisms, Release 10.1

```
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:**

- var – a variable name (default: x)

**EXAMPLES:**

```
sage: K.<u> = GF(7^10).over(GF(7^2))
sage: 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:

```
sage: K.modulus('y')
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:**

```
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
```

(continues on next page)
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:

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:

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)
sage: K = GF(5^4).over(F)
sage: K.base()
Finite Field in z2 of size 5^2

In case of iterated extensions, the base is itself an extension:

7.1. Extension of rings
sage: L = GF(5^8).over(K)
sage: L.base()
Field in z4 with defining polynomial x^2 + (4*z2 + 3)*x + z2 over its base
sage: L.base() is K
True

See also:

bases(), absolute_base(), is_defined_over()

bases()

Return the list of successive bases of this extension (including itself).

EXAMPLES:

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()
[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: 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: E = GF(5^3).over()
sage: E.construction()


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: 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)
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

(continues on next page)
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))`
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: A = GF(5^4).over(GF(5^2))
sage: B = GF(5^12).over(A)
sage: A.degree(GF(5^2))
2
sage: B.degree(A)
3
sage: B.degree(GF(5^2))
6
```

Note that base must be an explicit base over which the extension has been defined (as listed by the method `bases()`):

```
sage: A.degree(GF(5))
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:**

```
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
```
If `base` is omitted, the degree is computed over the base of the extension:

```python
sage: K.degree_over()
2
sage: L.degree_over()
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))
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:**

```python
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()
sage: 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)
sage: K2
Fraction Field of Order in Number Field in a with defining polynomial x^2 - 5 over its base
```
Note that there is no coercion between $K_1$ and $K_2$:

```
sage: K1.has_coerce_map_from(K2)
False
sage: K2.has_coerce_map_from(K1)
False
```

We check that when the extension is a field, its fraction field does not change:

```
sage: K1.fraction_field() is K1
True
sage: K2.fraction_field() is K2
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: 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: K = GF(5^2).over()  # over GF(5)
sage: x = K.gen(); x
z2
```

Observe that the generator lives in the extension:
sage: x.parent()
Field in \mathbb{Z}_2 with defining polynomial \(x^2 + 4x + 2\) over its base
sage: x.parent() is K
True

**gens**

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
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)
sage: S.<x> = QQ[]
sage: T.<y> = S[]
sage: T.over(S).gens()
(y,)
sage: T.over(QQ).gens()
(y, x)
```

**hom**

Return the unique homomorphism from this extension to `codomain` that sends `self.gens()` to the entries of `im_gens` and induces the map `base_map` on the base ring.

**INPUT:**

- `im_gens` – the images of the generators of this extension
- `codomain` – the codomain of the homomorphism; if omitted, it is set to the smallest parent containing all the entries of `im_gens`
- `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
- `category` – the category of the resulting morphism
- `check` – a boolean (default: `True`); whether to verify that the images of generators extend to define a map (using only canonical coercions)

**EXAMPLES:**

```sage
sage: K.<a> = GF(5^2).over()  # over GF(5)
sage: L.<b> = GF(5^6).over(K)
```

We define (by hand) the relative Frobenius endomorphism of the extension \(L/K\):
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 $L.$hom([b^5]) fails:

```sage
sage: L.hom([b^5])
Traceback (most recent call last):
...
ValueError: images do not define a valid homomorphism
```

What we need is to specify a base map:

```sage
sage: FrobK = K.hom([a^5])
sage: FrobL = L.hom([b^5], base_map=FrobK)
sage: 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
```

As a shortcut, we may use the following construction:

```sage
sage: phi = L.hom([b^5, a^5])
sage: phi
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
sage: phi == FrobL
True
```

**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:**

```sage
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: B.is_defined_over(A)
True
```

(continues on next page)
Note that an extension is defined over itself:

```sage
sage: A.is_defined_over(A)
True
sage: A.is_defined_over(GF(5^4))
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:**

```sage
sage: K = GF(5^5).over()  # over GF(5)
sage: K.is_field()
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:**

```sage
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:

```
sage: L.is_finite_over()
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:**

```
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:

```
sage: L.is_free_over()
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:**

```
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:

```python
sage: A.<a> = GF(5^2).over()   # over GF(5)
sage: B.<b> = GF(5^4).over(A)
sage: C.<c> = GF(5^12).over(B)
sage: D.<d> = GF(5^24).over(C)
```

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 over 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 over Field in b with defining polynomial x^2 + (3 - a)*x + a 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^3*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:

7.1. Extension of rings
sage: K = GF(5^2).over()  # over GF(5)
sage: x = K.random_element(); x  # random
3 + z2

sage: x.parent()
Field in z2 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))
sage: A.relative_degree()
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)
Finite Field of size 5
sage: common_base(GF(5^3), GF(5^7), True)
(Finite Field of size 5, 3, 7)

sage: common_base(GF(5^3), GF(7^5), False)
Traceback (most recent call last):
... NotImplementedError: 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], [])
sage: tower_bases(T, True)
([Univariate Polynomial Ring in y over Univariate Polynomial Ring in x over Rational Field], [1])

sage: K.<a> = Qq(5^2)
sage: L.<w> = K.extension(x^3 - 5)
sage: tower_bases(L, True)
([5-adic Eisenstein Extension Field in w defined by x^3 - 5 over its base field, 5-adic Unramified Extension Field in a defined by x^2 + 4*x + 2, 5-adic Field with capped relative precision 20], [1, 3, 6])
```

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:

```python
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

Generic class for elements lying in ring extensions.

additive_order()

Return the additive order of this element.

EXAMPLES:

```python
sage: K.<a> = GF(5^4).over(GF(5^2))
```
```
sage: a.additive_order()
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: F = GF(5^2)
```
```
sage: K.<z> = GF(5^4).over(F)
```
```
sage: x = z^10
```
```
sage: x
(z2 + 2) + (3*z2 + 1)*z
```
```
sage: y = x.backend()
```
```
sage: y
4*z4^3 + 2*z4^2 + 4*z4 + 4
```
```
sage: y.parent()
Finite Field in z4 of size 5^4
```
**in_base()**

Return this element as an element of the base.

**EXAMPLES:**

```
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
z2 + 1
sage: y.parent()
Finite Field in z2 of size 5^2
```

When the element is not in the base, an error is raised:

```
sage: z.in_base()
Traceback (most recent call last):
...  
ValueError: z is not in the base
```

```
sage: S.<X> = F[]
sage: E = S.over(F)
sage: f = E(1)
sage: g = f.in_base()
sage: 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:**

```
sage: A.<x> = PolynomialRing(QQ)
sage: E = A.over(QQ)
sage: E(0).is_nilpotent()
True
sage: E(x).is_nilpotent()
False
```

**is_prime()**

Return whether this element is a prime element in this ring.

**EXAMPLES:**

```
sage: A.<x> = PolynomialRing(QQ)
sage: E = A.over(QQ)
sage: E(x^2+1).is_prime()
True
sage: E(x^2-1).is_prime()
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: 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:

```python
sage: K.<a> = GF(5^4).over(GF(5^2))
sage: a.multiplicative_order()
624
```

sqrt(extend=True, all=False, name=None)
Return a square root or all square roots of this element.

INPUT:

* extend – a boolean (default: True); if “True”, return a square root in an extension ring, if necessary. Otherwise, raise a ValueError if the root is not in the ring
* all – a boolean (default: False); if True, return all square roots of this element, instead of just one.
* name – Required when extend=True and self is not a square. This will be the name of the generator extension.

Note: The option extend = True is often not implemented.
EXAMPLES:

```python
sage: 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]
```

class `sage.rings.ring_extension_element.RingExtensionFractionFieldElement`

Bases: `RingExtensionElement`

A class for elements lying in fraction fields of ring extensions.

`denominator()`

Return the denominator of this element.

EXAMPLES:

```python
sage: R.<x> = ZZ[]
sage: A.<a> = ZZ.extension(x^2 - 2)
sage: OK = A.over()  # over ZZ
sage: K = OK.fraction_field()
sage: 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
```

The denominator is an element of the ring which was used to construct the fraction field:

```python
sage: denom.parent()
Order in Number Field in a with defining polynomial x^2 - 2 over its base
sage: denom.parent() is OK
True
```

`numerator()`

Return the numerator of this element.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: A.<a> = ZZ.extension(x^2 - 2)
sage: OK = A.over()  # over ZZ
sage: K = OK.fraction_field()
sage: 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
```
The numerator is an element of the ring which was used to construct the fraction field:

```sage
sage: num.parent()
Order in Number Field in a with defining polynomial x^2 - 2 over its base
sage: num.parent() is OK
True
```

```class sage.rings.ring_extension_element.RingExtensionWithBasisElement
Bases: RingExtensionElement
A class for elements lying in finite free extensions.
```

```charpoly(base=None, var='x')
Return the characteristic polynomial of this element over base.
```

**INPUT:**

- `base` – a commutative ring (which might be itself an extension) or `None`

**EXAMPLES:**

```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
```

We check that the charpoly has coefficients in the base ring:

```sage: chi.base_ring()
Field in a with defining polynomial x^3 + 3*x + 3 over its base
sage: chi.base_ring() is K
True
```

and that it annihilates u:

```sage: chi(u)
0
```

Similarly, one can compute the characteristic polynomial over F:

```sage: u.charpoly(F)
x^6 + x^4 + 2*x^3 + 3*x + 4
```

A different variable name can be specified:

```sage: u.charpoly(F, var='t')
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()
x^2 + (1 + 2*a + 3*a^2)*x + 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.charpoly(GF(5^2))
Traceback (most recent call last):
...
ValueError: not (explicitly) defined over Finite Field in z2 of size 5^2

matrix(base=None)

Return the matrix of the multiplication by this element (in the basis output by basis_over()).

INPUT:

- base – a commutative ring (which might be itself an extension) or None

EXAMPLES:

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
sage: b*u
(3 + 2*a^2) + (2 + 2*a - a^2)*b
sage: u.matrix(K)
\[
\begin{bmatrix}
2 + a + 3*a^2 & 3 + 3*a + a^2 \\
3 + 2*a^2 & 2 + 2*a - a^2 \\
\end{bmatrix}
\]
sage: u.matrix(GF(5))
\[
\begin{bmatrix}
2 1 3 3 3 1 \\
1 3 1 2 0 3 \\
2 3 3 1 3 0 \\
3 0 2 2 2 4 \\
4 2 0 3 0 2 \\
0 4 2 4 2 0 \\
\end{bmatrix}
\]

If base is omitted, it is set to its default which is the base of the extension:

sage: u.matrix()
\[
\begin{bmatrix}
2 + a + 3*a^2 & 3 + 3*a + a^2 \\
3 + 2*a^2 & 2 + 2*a - a^2 \\
\end{bmatrix}
\]

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))
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:
\begin{verbatim}
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(K); 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()
Field in a with defining polynomial x^3 + 3*x + 3 over its base
sage: chi.base_ring() is K
True

and that it annihilates u:

sage: chi(u)
0

Similarly, one can compute the minimal polynomial over F:

sage: u.minpoly(F)
x^6 + 4*x^5 + x^4 + 2*x^2 + 3

A different variable name can be specified:

sage: u.minpoly(F, var='t')
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()
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))
Traceback (most recent call last):
  ...
ValueError: not (explicitly) defined over Finite Field in z2 of size 5^2
\end{verbatim}

\texttt{norm(base=None)}

Return the norm of this element over \texttt{base}.

INPUT:

\begin{itemize}
  \item base – a commutative ring (which might be itself an extension) or \texttt{None}
\end{itemize}

EXAMPLES:

\begin{verbatim}
sage: F = GF(5)
sage: K.<a> = GF(5^3).over(F)
sage: L.<b> = GF(5^6).over(K)
\end{verbatim}
We check that the norm lives in the base ring:

```
sage: nr.parent()
Field in a with defining polynomial x^3 + 3*x + 3 over its base
sage: nr.parent() is K
True
```

Similarly, one can compute the norm over $F$:

```
sage: u.norm(F)
4
```

We check the transitivity of the norm:

```
sage: u.norm(F) == nr.norm(F)
True
```

If `base` is omitted, it is set to its default which is the base of the extension:

```
sage: u.norm()
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))
Traceback (most recent call last):
  ...
ValueError: not (explicitly) defined over Finite Field in z2 of size 5^2
```

`polynomial`(base=None, 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
```

(continues on next page)
When the base is $F$, we obtain a bivariate polynomial:

```sage
sage: P = u.polynomial(F); P
(-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
sage: L.gens(F)
(c, b)
sage: P(c, b) == u
True
```

Similarly, when the base is $\text{GF}(5)$, we get a trivariate polynomial:

```sage
sage: P = u.polynomial(GF(5)); P
-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
True
```

Different variable names can be specified:

```sage
sage: u.polynomial(GF(5), var='y')
-y0^2*y1*y2 - y0^2*y2 - y0*y1*y2 - y0^2 + y0*y2 - y1*y2 + 2*y0 + 2 -
˓→2
sage: u.polynomial(GF(5), var=['x', 'y', 'z'])
-x^2*y*z - x^2*z - x*y*z - x^2 + x*y - 2*x^2*z - y^2 - 2*x - y + 2
```

If base is omitted, it is set to its default which is the base of the extension:

```sage
sage: u.polynomial()
((-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
sage: u.polynomial(GF(5^3))
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
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)
```

(continues on next page)
sage: tr = u.trace(K); tr
-1 + 3*a + 2*a^2

We check that the trace lives in the base ring:

sage: tr.parent()
Field in a with defining polynomial x^3 + 3*x + 3 over its base
sage: tr.parent() is K
True

Similarly, one can compute the trace over F:

sage: u.trace(F)
0

We check the transitivity of the trace:

sage: u.trace(F) == tr.trace(F)
True

If base is omitted, it is set to its default which is the base of the extension:

sage: u.trace()
-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))
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: F = GF(5)
sage: K.<a> = GF(5^2).over()  # over F
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()
(-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))
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))
sage: V, i, j = K.free_module()
sage: i.is_injective()
True
```

**is_surjective()**

Return whether this morphism is surjective.

**EXAMPLES:**

```
sage: K = GF(11^6).over(GF(11^3))
sage: V, i, j = K.free_module()
sage: i.is_surjective()
True
```

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: K = GF(11^6).over(GF(11^3))
sage: V, i, j = K.free_module()
sage: j.is_injective()
True

is_surjective()
Return whether this morphism is injective.

EXAMPLES:

sage: K = GF(11^6).over(GF(11^3))
sage: V, i, j = K.free_module()
sage: j.is_surjective()
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:

sage: F = GF(5)
sage: K.<a> = GF(5^2).over(F)
sage: L.<b> = GF(5^6).over(K)
We define the absolute Frobenius of L:

sage: FrobL = L.hom([b^5, a^5])
sage: 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
sage: 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:

7.3. Morphisms between extension of rings
is_identity()

Return whether this morphism is the identity.

EXAMPLES:

```
sage: K.<a> = GF(5^2).over()  # over GF(5)
sage: FrobK = K.hom([a^5])
sage: FrobK.is_identity()
False
sage: (FrobK^2).is_identity()
True
```

Coercion maps are not considered as identity morphisms:

```
sage: L.<b> = GF(5^6).over(K)
sage: iota = L.defining_morphism()
sage: iota
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 |--> a
sage: iota.is_identity()
False
```

is_injective()

Return whether this morphism is injective.

EXAMPLES:

```
sage: K = GF(5^10).over(GF(5^5))
sage: iota = K.defining_morphism()
sage: 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()
sage: 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: K = GF(5^10).over(GF(5^5))
sage: iota = K.defining_morphism()
sage: 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()
sage: iota
Ring morphism:
  From: Integer Ring
  To:   Finite Field of size 7 over its base
    Defn: 1 |--> 1
sage: iota.is_surjective()
True
```

7.3. Morphisms between extension of rings
CHAPTER EIGHT

GENERIC DATA STRUCTURES AND ALGORITHMS FOR RINGS

8.1 Generic data structures and algorithms for rings

AUTHORS:

• Lorenz Panny (2022): ProductTree, prod_with_derivative()

class sage.rings.generic.ProductTree(leaves)

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
sage: F = GF(65537)
sage: a = F(1111)
sage: assert a.multiplicative_order() == 1024
sage: R.<x> = F[]
sage: ms = [x - a^i for i in range(1024)] # roots of unity
sage: ys = [F.random_element() for _ in range(1024)] # input vector
sage: zs = ProductTree(ms).remainders(R(ys)) # compute FFT!
sage: 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:**

```
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.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.root().factor()
2 * 3 * 5 * 7 * 11 * 13 * 17 * 19 * 23 * 29 * 31 * 37 * 41 * 43 * 47 * 53 * 59 * 61...
˓→ * 67 * 71 * 73 * 79 * 83 * 89 * 97
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]

We can access the individual layers of the tree:

sage: tree.layers
[(2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,...
˓→ 79, 83, 89, 97), (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: 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[]
sage: fs = [x^2 + 2*x + 3, 4*x + 5, 6*x^7 + 8*x + 9]
sage: prod(fs)
24*x^10 + 78*x^9 + 132*x^8 + 90*x^7 + 32*x^6 + 140*x^5 + 293*x^4 + 218*x^3 + 135
sage: prod(fs).derivative()
240*x^9 + 702*x^8 + 1056*x^7 + 630*x^6 + 128*x^5 + 420*x^4 + 586*x^3 + 318
sage: F, df = prod_with_derivative((f, f.derivative()) for f in fs)
sage: F
24*x^10 + 78*x^9 + 132*x^8 + 90*x^7 + 32*x^6 + 140*x^5 + 293*x^4 + 218*x^3 + 135
sage: df
240*x^9 + 702*x^8 + 1056*x^7 + 630*x^6 + 128*x^5 + 420*x^4 + 586*x^3 + 318
```

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

`sage.rings.big_oh.O(*x, **kwds)`

Big O constructor for various types.

EXAMPLES:

This is useful for writing power series elements:

```
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)
```

A power series ring is created implicitly if a polynomial element is passed:

```
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
```

This is also useful to create $p$-adic numbers:

```
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)
```

It behaves well with respect to adding negative powers of $p$: 
\begin{verbatim}
sage: a = O(11^-32); a
   # needs sage.rings.padics
0(11^-32)
sage: a.parent()
   # needs sage.rings.padics
11-adic Field with capped relative precision 20

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 + O(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 + O(11^15)
sage: K(11^-12, absprec=15)
11^-12 + O(11^15)
sage: K(11^-12, 15)
11^-12 + O(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) + 0(y^(1/3))
y^(1/5) + 0(y^(1/3))
\end{verbatim}
\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: 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: 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:

```
sage: unsigned_oo + 5
Infinity
```

We make \(1/\text{unsigned}_\text{oo}\) return the integer 0 so that arithmetic of the following type works:

```
sage: (1/unsigned_oo) + 2
2
sage: 32/5 - (2.439/unsigned_oo)
32/5
```

Note that many operations are not defined, since the result is not well-defined:

```
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.

```
sage: 0/unsigned_oo
0
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.

```
sage: P = InfinityRing; P
The Infinity Ring
sage: P(5)
A positive finite number
```

The symbol \(\text{oo}\) is predefined as a shorthand for \(+\text{Infinity}\):

```
sage: oo
+Infinity
```

We compare finite and infinite elements:
We can do more arithmetic than in the unsigned infinity ring:

\[
\begin{align*}
\text{sage: } & 2 \times \infty \\
& +\infty \\
\text{sage: } & -2 \times \infty \\
& -\infty \\
\text{sage: } & 1 - \infty \\
& -\infty \\
\text{sage: } & 1 / \infty \\
& 0 \\
\text{sage: } & -1 / \infty \\
& 0
\end{align*}
\]

We make \(1 / \infty\) and \(1 / -\infty\) return the integer 0 instead of the infinity ring Zero so that arithmetic of the following type works:

\[
\begin{align*}
\text{sage: } & (1/\infty) + 2 \\
& 2 \\
\text{sage: } & 32/5 - (2.439/-\infty) \\
& 32/5
\end{align*}
\]

If we try to subtract infinities or multiply infinity by zero we still get an error:

\[
\begin{align*}
\text{sage: } & \infty - \infty \\
\text{Traceback (most recent call last):} \\
& \ldots \\
& \text{SignError: cannot add infinity to minus infinity} \\
\text{sage: } & 0 * \infty \\
\text{Traceback (most recent call last):} \\
& \ldots \\
& \text{SignError: cannot multiply infinity by zero} \\
\text{sage: } & P(2) + P(-3) \\
\text{Traceback (most recent call last):} \\
& \ldots \\
& \text{SignError: cannot add positive finite value to negative finite value}
\end{align*}
\]

Signed infinity can also be represented by RR / RDF elements. But unsigned infinity cannot:

\[
\begin{align*}
\text{sage: } & \infty \text{ in } \text{RR}, \infty \text{ in } \text{RDF} \\
& (\text{True}, \text{True}) \\
\text{sage: } & \text{unsigned}_\infty \text{ in } \text{RR}, \text{unsigned}_\infty \text{ in } \text{RDF} \\
& (\text{False}, \text{False})
\end{align*}
\]

```python
class sage.rings.infinity.AnInfinity
    Bases: object
```

9.2. Signed and Unsigned Infinities
\texttt{lcm(x)}

Return the least common multiple of \texttt{oo} and \texttt{x}, which is by definition \texttt{oo} unless \texttt{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} \texttt{sage.rings.infinity.FiniteNumber(parent, x)}

\textbf{Bases:} \texttt{RingElement}

Initialize self.

\textbf{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}

\textbf{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} \texttt{sage.rings.infinity.InfinityRing_class}

\textbf{Bases:} \texttt{Singleton, Ring}

Initialize self.

\textbf{fraction_field()}

This isn’t really a ring, let alone an integral domain.

\textbf{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

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

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

exception sage.rings.infinity.SignError
    Bases: ArithmeticError
    Sign error exception.

class sage.rings.infinity.UnsignedInfinity(*args)
    Bases: _uniq, AnInfinity, InfinityElement
    Initialize self.

class sage.rings.infinity.UnsignedInfinityRing_class
    Bases: Singleton, Ring
    Initialize self.
```
**fraction_field()**

The unsigned infinity ring isn’t an integral domain.

EXAMPLES:

```
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:

```
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:

```
sage: UnsignedInfinityRing.gens()
[Infinity]
```

**less_than_infinity()**

This is the element that represents a finite value.

EXAMPLES:

```
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:

```
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, RR, RealField(200), RDF, RLF, RIF]
sage: for R in rings:
    ....:     print('testing {}'.format(R))
    ....:     test_comparison(R)

testing Integer Ring
testing Rational Field
testing Real Field with 53 bits of precision
testing Real Field with 200 bits of precision
testing Real Double Field
testing Real Lazy Field
testing Real Interval Field with 53 bits of precision
sage: test_comparison(AA)  
˓→optional - sage.rings.number_field

Comparison with number fields does not work:

sage: K.<sqrt3> = NumberField(x^2 - 3)     # known bug
˓→optional - sage.rings.number_field
sage: (-oo < 1 + sqrt3) and (1 + sqrt3 < oo)  # known bug
˓→optional - sage.rings.number_field
False

The symbolic ring handles its own infinities, but answers False (meaning: cannot decide) already for some very elementary comparisons:

sage: test_comparison(SR)  
˓→optional - sage.symbolic

(continues on next page)
sage.rings.infinity.test_signed_infinity(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:

```
sage: InfinityRing(float('+inf'))
+Infinity
sage: InfinityRing(float('-inf'))
-Infinity
sage: oo > float('+inf')
False
sage: oo == float('+inf')
True
```

EXAMPLES:

```
sage: from sage.rings.infinity import test_signed_infinity
sage: for pos_inf in [oo, float('+inf'), RLF(oo), RIF(oo), SR(oo)]:
    ....:     test_signed_infinity(pos_inf)
```

9.3 Support Python’s numbers abstract base class

See also:

PEP 3141 for more information about numbers.

sage.rings.numbers_abc.register_sage_classes()

Register all relevant Sage classes in the numbers hierarchy.

EXAMPLES:

```
sage: import numbers
sage: isinstance(5, numbers.Integral)
True
sage: isinstance(5, numbers.Number)
True
sage: isinstance(5/1, numbers.Integral)
False
sage: isinstance(22/7, numbers.Rational)
```
True
sage: isinstance(1.3, numbers.Real)
   True
sage: isinstance(CC(1.3), numbers.Real)
   False
sage: isinstance(CC(1.3 + I), numbers.Complex)
   True
sage: isinstance(RDF(1.3), numbers.Real)
   True
sage: isinstance(CDF(1.3, 4), numbers.Complex)
   True
sage: isinstance(AA(sqrt(2)), numbers.Real)
   True
sage: isinstance(QQbar(I), numbers.Complex)
   True

This doesn’t work with symbolic expressions at all:
sage: isinstance(pi, numbers.Real)
   False
sage: isinstance(I, numbers.Complex)
   False
sage: isinstance(sqrt(2), numbers.Real)
   False

Because we do this, NumPy’s isscalar() recognizes Sage types:
sage: from numpy import isscalar
   #˓→optional - numpy
sage: isscalar(3.141)
   #˓→optional - numpy
   True
sage: isscalar(4/17)
   #˓→optional - numpy
   True
10.1 Derivations

Let $A$ be a ring and $B$ be a 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[]
sage: M = A.derivation_module()
sage: M
Module of derivations over Multivariate Polynomial Ring in x, y, z over Rational Field
```

The method `gens()` returns the generators of this module:

```
sage: A.<x,y,z> = QQ[]
sage: M = A.derivation_module()
sage: M.gens()
(d/dx, d/dy, d/dz)
```

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)
sage: d
2*d/dx + z*d/dy + (x^2 + y^2)*d/dz
```

and now play with them:
Alternatively we can use the method \texttt{derivation()} of the ring \( A \) to create derivations:

\begin{verbatim}
sage: Dx = A.derivation(x); Dx
d/dx
sage: Dy = A.derivation(y); Dy
d/dy
sage: Dz = A.derivation(z); Dz
d/dz
sage: A.derivation([2, z, x^2+y^2])
2*d/dx + z*d/dy + (x^2 + y^2)*d/dz
\end{verbatim}

Sage knows moreover that \( M \) is a Lie algebra:

\begin{verbatim}
sage: M.category()
Join of Category of lie algebras with basis over Rational Field and
Category of modules with basis over
Multivariate Polynomial Ring in x, y, z over Rational Field
\end{verbatim}

Computations of Lie brackets are implemented as well:

\begin{verbatim}
sage: Dx.bracket(Dy)
0
sage: d.bracket(Dx)
-2*x*d/dz
\end{verbatim}

At the creation of a module of derivations, a codomain can be specified:

\begin{verbatim}
sage: B = A.fraction_field()
sage: A.derivation_module(B)
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
\end{verbatim}

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:

\begin{verbatim}
sage: ev = A.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
\end{verbatim}
Now we use this ring homomorphism to define a structure of $A$-algebra on $\mathbb{Q}$ and then build the following module of derivations:

```sage
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 $(0,1,2)$:

```sage
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:

```sage
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:

```sage
sage: d = M.gen(); d
[x |--> y, y |--> z, z |--> x] - id
sage: d(x)
-x + y
sage: d(y)
-y + z
sage: d(z)
x - z
sage: d(x + y + z)
0
```

10.1. Derivations
Author:

- Xavier Caruso (2018-09)

Class sage.rings.derivation.RingDerivation

Bases: ModuleElement

An abstract class for twisted and untwisted derivations over commutative rings.

codomain()

Return the codomain of this derivation.

Examples:

```sage
codomain()

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

codomain()

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
```

domain()

Return the domain of this derivation.

Examples:

```sage
domain()

sage: R.<x,y> = QQ[]
sage: f = R.derivation(y); f
d/dy
sage: f.domain()
Multivariate Polynomial Ring in x, y over Rational Field
sage: f.domain() is R
True
```

class sage.rings.derivation.RingDerivationModule(domain, codomain, twist=None)

Bases: Module, UniqueRepresentation

A class for modules of derivations over a commutative ring.

basis()

Return a basis of this module of derivations.

Examples:

```sage
class sage.rings.derivation.RingDerivationModule(domain, codomain, twist=None)

basis()

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

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

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 \(d_i(x_j) = 1 \text{ if } i = j \text{ and } 0 \text{ otherwise.} \)
and $d_i(x_j) = 0$ if $i \neq j$.

**EXAMPLES:**

```
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:**

```
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:**

```
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 $\text{twist} - \text{id}$:

```
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 |--> y, y |--> x)
sage: M.gens()
([x |--> y, y |--> x] - id,)
```

**ngens()**

Return the number of generators of this module of derivations.

**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.ngens()
2

Indeed, generators are:

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 \(\text{twist - id}\):

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 |--> y, y |--> x)
sage: M.ngens()
1
sage: M.gen()
[x |--> y, y |--> x] - id

random_element(*args, **kwds)

Return a random derivation in this module.

EXAMPLES:

sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module()
sage: M.random_element()
# random
(x^2 + x*y - 3*y^2 + x + 1)*d/dx + (-2*x^2 + 3*x*y + 10*y^2 + 2*x + 8)*d/dy

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.

EXAMPLES:

sage: R.<x,y> = QQ[]
sage: M = R.derivation_module()
sage: M.basis()
Family (d/dx, d/dy)
sage: M.ring_of_constants()
Rational Field

some_elements()

Return a list of elements of this module.

EXAMPLES:
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module()
sage: M.some_elements()
[d/dx, d/dy, x*d/dx, x*d/dy, y*d/dx, y*d/dy]

twisting_morphism()
Return the twisting homomorphism of the derivations in this module.

EXAMPLES:

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:

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:

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()
sage: D
x*([x |--> y, y |--> x] - id)
sage: D.domain()
Fraction Field of Multivariate Polynomial Ring in x, y over Integer Ring

sage: D(1/x)
(x - y)/y

list()
Return the list of coefficient of this twisted derivation on the canonical basis.

EXAMPLES:
**General Rings, Ideals, and Morphisms, Release 10.1**

```python
sage: R.<x,y> = QQ[]
sage: K = R.fraction_field()
sage: theta = K.hom([y,x])
sage: M = K.derivation_module(twist=theta)
sage: M.basis()
Family (twisting_morphism - id,)
sage: f = (x+y) * M.gen()
sage: f
(x + y)*(twisting_morphism - id)
sage: f.list()
[x + y]
```

**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:**

```python
sage: R.<x,y> = ZZ[]
sage: theta = R.hom([y,x])
sage: D = R.derivation(x, twist=theta); D
x*(x |--> y, y |--> x) - id)
sage: f = R.hom([x^2, y^3])
sage: g = D.precompose(f); g
x^2*(x |--> y^3, y |--> x^2) - [x |--> x^2, y |--> y^3])
```

Observe that the $g$ is no longer a $\theta$-derivation but a $(\theta \circ f)$-derivation:

```python
sage: g.parent().twisting_morphism()
Ring endomorphism of Multivariate Polynomial Ring in x, y over Integer Ring
  Defn: x |--> y^2
  y |--> x^3
```

**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:**

```python
sage: R.<x,y> = ZZ[]
sage: theta = R.hom([y,x])
sage: D = R.derivation(x, twist=theta); D
x*(x |--> y, y |--> x) - id)
sage: f = R.hom([x^2, y^3])
sage: g = D.postcompose(f); g
x^2*(x |--> y^3, y |--> x^2) - [x |--> x^2, y |--> y^3])
```

10.1. Derivations 183
Observe that the $g$ is no longer a $\theta$-derivation but a $(f \circ \theta)$-derivation:

```
sage: g.parent().twisting_morphism()
Ring endomorphism of Multivariate Polynomial Ring in x, y over Integer Ring
  Defn: x |--> y^3
    y |--> x^2
```

```python
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:

```
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:

```
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()
```

(continues on next page)
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.

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}
```

postcompose(morphism)

Return the derivation obtained by applying first this 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 coerces

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 ev:
```

```
sage: dx = Dx.postcompose(ev)
sage: dy = Dy.postcompose(ev)
sage: f = x^2 + y^2
sage: dx(f)
0
```
sage: dy(f)
2

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:

sage: D.precompose(D)
Traceback (most recent call last):
  ...  
TypeError: you must give an homomorphism of rings
pth_power()

Return 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:

```sage
sage: R.<x,y> = GF(5)[]
         # optional - sage.rings.finite_rings
sage: Dx = R.derivation(x)
         # optional - sage.rings.finite_rings
sage: Dx.pth_power()  # optional - sage.rings.finite_rings
0
sage: (x*Dx).pth_power()  # optional - sage.rings.finite_rings
x*d/dx
sage: (x^6*Dx).pth_power()  # optional - sage.rings.finite_rings
x^26*d/dx
sage: Dy = R.derivation(y)  # optional - sage.rings.finite_rings
sage: (x*Dx + y*Dy).pth_power()  # optional - sage.rings.finite_rings
x*d/dx + y*d/dy
```

An error is raised if the domain has characteristic zero:

```sage
sage: R.<x,y> = QQ[]
sage: Dx = R.derivation(x)
sage: Dx.pth_power()  # 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:

```sage
sage: R.<x,y> = Integers(10)[]
sage: Dx = R.derivation(x)
sage: Dx.pth_power()  # Traceback (most recent call last):
  ...  TypeError: the domain of the derivation must have positive and prime
         characteristic
```

class **sage.rings.derivation.RingDerivationWithoutTwist_fraction_field**(parent, arg=None)

Bases: **RingDerivationWithoutTwist_wrapper**

This class handles derivations over fraction fields.

class **sage.rings.derivation.RingDerivationWithoutTwist_function**(parent, arg=None)

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:**

```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:**

```python
sage: R.<x,y> = GF(5)[[]]
# optional - sage.rings.finite_rings
sage: M = R.derivation_module()
# optional - sage.rings.finite_rings
sage: M.basis()
# optional - sage.rings.finite_rings
Family (d/dx, d/dy)
sage: R.derivation(x).list()
# optional - sage.rings.finite_rings
[1, 0]
sage: R.derivation(y).list()
# optional - sage.rings.finite_rings
[0, 1]
sage: f = x*R.derivation(x) + y*R.derivation(y); f
x*d/dx + y*d/dy
sage: f.list()
# optional - sage.rings.finite_rings
[x, y]
```

**class** `sage.rings.derivation.RingDerivationWithoutTwist_quotient` *(parent, arg=None)*

**Bases:** `RingDerivationWithoutTwist_wrapper`

This class handles derivations over quotient rings.

**class** `sage.rings.derivation.RingDerivationWithoutTwist_wrapper` *(parent, arg=None)*

**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: R.<X,Y> = GF(5)[]
      #optional - sage.rings.finite_rings
sage: S.<x,y> = R.quo([X^5, Y^5])
      #optional - sage.rings.finite_rings
sage: M = S.derivation_module()
      #optional - sage.rings.finite_rings
sage: M.basis()
      #optional - sage.rings.finite_rings
Family (d/dx, d/dy)
sage: S.derivation(x).list()
      #optional - sage.rings.finite_rings
[1, 0]
sage: S.derivation(y).list()
      #optional - sage.rings.finite_rings
[0, 1]
sage: f = x*S.derivation(x) + y*S.derivation(y); f
      #optional - sage.rings.finite_rings
x*d/dx + y*d/dy
sage: f.list()
      #optional - sage.rings.finite_rings
[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()
[]
```
INDEXES AND TABLES

• Index
• Module Index
• Search Page
sage.rings.abc, 26
sage.rings.big_oh, 163
sage.rings.derivation, 175
sage.rings.fraction_field, 99
sage.rings.fraction_field_element, 105
sage.rings.generic, 159
sage.rings.homset, 76
sage.rings.ideal, 35
sage.rings.ideal_monoid, 52
sage.rings.infinity, 165
sage.rings.localization, 111
sage.rings.morphism, 55
sage.rings.noncommutative_ideals, 52
sage.rings.numbers_abc, 173
sage.rings.quotient_ring, 79
sage.rings.quotient_ring_element, 94
sage.rings.ring, 1
sage.rings.ring_extension, 123
sage.rings.ring_extension_element, 144
sage.rings.ring_extension_morphism, 154
INDEX

A

absolute_base() (sage.rings.ring_extension.RingExtension_generic method), 130
absolute_degree() (sage.rings.ring_extension.RingExtension_generic method), 131
absolute_norm() (sage.rings.ideal.Ideal_generic method), 38
additive_order() (sage.rings.ring_extension_element.RingExtensionElement method), 144
Algebra (class in sage.rings.ring), 2
algebraic_closure() (sage.rings.ring.Field method), 11
AlgebraicField (class in sage.rings.abc), 26
AlgebraicField_common (class in sage.rings.abc), 26
AlgebraicRealField (class in sage.rings.abc), 26
ambient() (sage.rings.quotient_ring.QuotientRing_nc method), 85
AnInfinity (class in sage.rings.infinity), 167
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), 131
backend() (sage.rings.ring_extension_element.RingExtensionElement method), 144
base() (sage.rings.ring_extension.RingExtension_generic method), 131
base_extend() (sage.rings.ring.Ring method), 17
base_map() (sage.rings.morphism.RingHomomorphism_im_gens method), 73
base_map() (sage.rings.ring_extension_morphism.RingExtensionHomomorphism method), 155
base_ring() (sage.rings.fraction_field.FractionField_generic method), 102
base_ring() (sage.rings.ideal.Ideal_generic method), 39
bases() (sage.rings.ring_extension.RingExtension_generic method), 132
basis() (sage.rings.derivation.RingDerivationModule method), 178
basis_over() (sage.rings.ring_extension.RingExtensionWithBasis method), 125

C

CallableSymbolicExpressionRing (class in sage.rings.abc), 27
category() (sage.rings.ideal.Ideal_generic method), 40
category() (sage.rings.ring.Ring method), 17
characteristic() (sage.rings.fraction_field.FractionField_generic method), 102
characteristic() (sage.rings.localization.Localization method), 116
characteristic() (sage.rings.quotient_ring.QuotientRing_element method), 85
characteristic() (sage.rings.ring.Algebra method), 2
characteristic() (sage.rings.ring_extension.RingExtension_generic method), 132
charpoly() (sage.rings.ring_extension_element.RingExtensionWithBasisElement method), 148
class_group() (sage.rings.ring.PrincipalIdealDomain method), 14
class_number() (sage.rings.fraction_field.FractionField_1poly_field method), 101
codomain() (sage.rings.derivation.RingDerivation_module method), 178
codomain() (sage.rings.derivation.RingDerivationModule method), 178
common_base() (in module sage.rings.ring_extension), 142
CommutativeAlgebra (class in sage.rings.ring), 3
CommutativeRing (class in sage.rings.ring), 3
ComplexBallField (class in sage.rings.abc), 27
ComplexDoubleField (class in sage.rings.abc), 28
ComplexField (class in sage.rings.abc), 28
ComplexIntervalField (class in sage.rings.abc), 28
classification() (sage.rings.fraction_field.FractionField_generic method), 102
classification() (sage.rings.quotient_ring.QuotientRing_element method), 85
construction() (sage.rings.ring_extension.RingExtension method), 133
content() (sage.rings.ring.PrincipalIdealDomain method), 15
cover() (sage.rings.quotient_ring.QuotientRing_nc method), 86
cover_ring() (sage.rings.quotient_ring.QuotientRing_nc method), 86
create_key_and_extra_args()
(sage.rings.ring_extension.RingExtensionFactory method), 124
create_object() (sage.rings.ring_extension.RingExtensionFactory method), 125
Cyclic() (in module sage.rings.ideal), 35

D
DedekindDomain (class in sage.rings.ring), 8
defining_ideal() (sage.rings.quotient_ring.QuotientRing_nc method), 87
defining_morphism()
(sage.rings.derivation.RingDerivationModule method), 179
defining_morphism()
(sage.rings.ring_extension.RingExtension_generic method), 133
degree() (sage.rings.ring_extension.RingExtension method), 134
degree_over() (sage.rings.ring_extension.RingExtension_generic method), 134
denominator() (sage.rings.fraction_field_element.FractionFieldElement method), 118
denominator() (sage.rings.localization.LocalizationElement method), 147
denominator() (sage.rings.ring_extension_element.RingExtensionElement method), 147

derivation() (sage.rings.ring.CommutativeRing method), 3
derivation_module()
(sage.rings.ring.CommutativeRing method), 4
divides() (sage.rings.ideal.Ideal_principal method), 49
divides() (sage.rings.ring.Field method), 11
domain() (sage.rings.derivation.RingDerivation method), 178
domain() (sage.rings.derivation.RingDerivationModule method), 179
dual_basis() (sage.rings.derivation.RingDerivationModule method), 179

E
Element (sage.rings.homset.RingHomset_generic attribute), 76
Element (sage.rings.homset.RingHomset_quo_ring attribute), 77

F
factor() (sage.rings.localization.LocalizationElement method), 119
Field (class in sage.rings.ring), 11
FiniteDomain (in module sage.rings.ideal), 36
FiniteNumber (class in sage.rings.infinity), 168
fraction_field() (sage.rings.infinity.InfinityRing_class method), 168
fraction_field() (sage.rings.infinity.UnsignedInfinityRing_class method), 170
fraction_field() (sage.rings.localization.Localization method), 116
fraction_field() (sage.rings.ring.CommutativeRing method), 6
fraction_field() (sage.rings.ring.Field method), 11
fraction_field() (sage.rings.ring_extension.RingExtension_generic method), 135
fraction_field() (sage.rings.ring_extension.RingExtensionWithBasis method), 126
fraction_field() (sage.rings.ring_extension.RingExtensionWithGen method), 128
FractionField() (in module sage.rings.fraction_field), 99
FractionFieldpopulate_element() (in sage.rings.fraction_field), 101
FractionField_generic (class in sage.rings.fraction_field), 102
FractionFieldElement (class in sage.rings.fraction_field_element), 105
FractionFieldElement_lpoly_field (class in sage.rings.fraction_field_element), 108
gens_reduced() (sage.rings.ideal.Ideal_generic method), 41
graded_free_resolution() (sage.rings.ideal.Ideal_generic method), 41
free_module() (sage.rings.ring_extension.RingExtensionWithBasis method), 127
free_resolution() (sage.rings.ideal.Ideal_generic method), 40
frobenius_endomorphism() (sage.rings.ring.CommutativeRing method), 7
FrobeniusEndomorphism_generic (class in sage.rings.morphism), 61
from_base_ring() (sage.rings.ring_extension.RingExtension_generic method), 136
function_field() (sage.rings.fraction_field.FractionField_1poly_field method), 101
G
gcd() (sage.rings.ideal.Ideal_pid method), 47
gcd() (sage.rings.ring.PrincipalIdealDomain method), 15
gen() (sage.rings.derivation.RingDerivationModule method), 180
gen() (sage.rings.fraction_field.FractionField_generic method), 103
gen() (sage.rings.ideal.Ideal_generic method), 40
gen() (sage.rings.ideal.Ideal_principal method), 49
gen() (sage.rings.infinity.InfinityRing_class method), 168
gen() (sage.rings.infinity.UnsignedInfinityRing_class method), 171
gen() (sage.rings.localization.Localization method), 117
gen() (sage.rings.quotient_ring.QuotientRing_nc method), 87
gen() (sage.rings.ring_extension.RingExtension_generic method), 136
generators() (in module sage.rings.ring_extension), 142
gens() (sage.rings.derivation.RingDerivationModule method), 180
gens() (sage.rings.ideal.Ideal_generic method), 41
gen() (sage.rings.infinity.InfinityRing_class method), 169
gen() (sage.rings.infinity.UnsignedInfinityRing_class method), 171
gen() (sage.rings.localization.Localization method), 117
gen() (sage.rings.ring_extension.RingExtension_generic method), 137
gen() (sage.rings.ring_extension.RingExtensionWithGen method), 129

table
is_surjective() (sage.rings.fraction_field.FractionField_class method), 100
is_surjective() (sage.rings.morphism.RingHomomorphism method), 67
is_surjective() (sage.rings.quotient_ring_element.QuotientRingElement method), 90
is_surjective() (sage.rings.ring_extension_morphism.MapRingHomomorphism method), 91
is_trivial() (sage.rings.ideal.Ideal_generic method), 44
is_unit() (sage.rings.localization.LocalizationElement method), 119
is_unit() (sage.rings.quotient_ring_element.QuotientRingElement method), 95
is_unit() (sage.rings.ring_extension_element.RingExtensionElement method), 146
is_zero() (sage.rings.derivation.RingDerivationWithoutTwist.localization (class in sage.rings.localization), 114
is_zero() (sage.rings.derivation.RingDerivationWithoutTwist.localization() (sage.rings.ring.CommutativeRing LocalizationElement (class in sage.rings.derivation.RingDerivationWithoutTwist_function method), 8
is_zero() (sage.rings.derivation.RingDerivationWithoutTwist_zero sage.rings.localization), 118
is_zero() (sage.rings.fraction_field_element.FractionFieldElement method), 107
K
Katsura() (in module sage.rings.ideal), 50
kernel() (sage.rings.morphism.RingHomomorphism method), 67
kernel() (sage.rings.morphism.RingHomomorphism.cover method), 69
krull_dimension() (sage.rings.localization.Localization matrix() (sage.rings.ring_extension_element.RingExtensionWithBasisElement method), 118
krull_dimension() (sage.rings.ring.CommutativeRing method), 8
krull_dimension() (sage.rings.ring.DedekindDomain method), 10
krull_dimension() (sage.rings.ring.Field method), 12
L
lc() (sage.rings.quotient_ring_element.QuotientRingElement method), 95
lcm() (sage.rings.infinity.AnInfinity method), 167
leaves() (sage.rings.generic.ProductTree method), 160
less_than_infinity() (sage.rings.infinity.UnsignedInfinityRing_class method), 171
LessThanInfinity (class in sage.rings.infinity), 169
lift() (sage.rings.morphism.RingHomomorphism method), 68
make_element() (in module sage.rings.fraction_field_element), 109
make_element_old() (in module sage.rings.fraction_field_element), 109
MapFreeModuleToRelativeRing (class in sage.rings.ring_extension_morphism), 154
MapRelativeRingToFreeModule (class in sage.rings.ring_extension_morphism), 154
maximal_order() (sage.rings.fraction_field.FractionField_Ipoly_field method), 149
minimal-associated_primes() (sage.rings.ideal.Ideal_generic method), 44
minpoly() (sage.rings.ring_extension_element.RingExtensionWithBasisElement method), 149
MinusInfinity (class in sage.rings.infinity), 170
module
sage.rings.abc, 26
sage.rings.big_oh, 163
sage.rings.derivation, 175
sage.rings.fraction_field, 99
sage.rings.fraction_field_element, 105
sage.rings.generic, 159
sage.rings.homset, 76
sage.rings.ideal, 35
modulus() (sage.rings.ring_extension.RingExtensionWithGen method), 130
monomial_coefficients() (sage.rings.derivation.RingDerivationWithoutTwist method), 183
monomials() (sage.rings.quotient_ring_element.RingExtensionFractionFieldElement
method), 104
morphism_from_cover() (sage.rings.morphism.RingHomomorphism_from_quotient
method), 73
multiplicative_order() (sage.rings.ring_extension_element.RingExtensionElement
method), 146

natural_map() (sage.rings.homset.RingHomset_generic method), 76
ngens() (sage.rings.derivation.RingDerivationModule method), 180
ngens() (sage.rings.ring_extension.RingExtension_generic method), 140
ngens() (sage.rings.ideal.Ideal_generic method), 44
ngens() (sage.rings.infinity.InfinityRing_class method), 169
ngens() (sage.rings.infinity.UnsignedInfinityRing_class method), 171
ngens() (sage.rings.localization.Localization method), 118
ngens() (sage.rings.quotient_ring_quotientRing_nc method), 92
ngens() (sage.rings.ring_extension.RingExtension_generic method), 140
NoetherianRing (class in sage.rings.ring), 14
norm() (sage.rings.ideal.Ideal_generic method), 45
norm() (sage.rings.ring_extension_element.RingExtensionElement method), 150
normalize_extra_units() (in module sage.rings.localization), 120
nth_root() (sage.rings.ring_extension_element.RingExtensionElement mtod), 107
NumberField_cyclotomic (class in sage.rings.abc), 29
NumberField_quadratic (class in sage.rings.abc), 30
numerator() (sage.rings.ring_extension_element.RingExtensionElement
method), 175
numerator() (sage.rings.ring_extension_element.RingExtensionElement
method), 120
numerator() (sage.rings.ring_extension_element.RingExtensionElement
method), 147

O
O (in module sage.rings.big_oh), 163
one() (sage.rings.ring.Ring method), 22
Order (class in sage.rings.abc), 30
order() (sage.rings.ring.Ring method), 23

P
pAdicField (class in sage.rings.abc), 33
pAdicRing (class in sage.rings.abc), 33
parameter() (sage.rings.ring.EuclideanDomain
method), 11
PlusInfinity (class in sage.rings.infinity), 170
polynomial() (sage.rings.ring_extension_element.RingExtensionElement
method), 174
postcompose() (sage.rings.derivation.RingDerivationWithoutTwist
method), 185
postcompose() (sage.rings.derivation.RingDerivationWithTwist_generic
method), 183
power() (sage.rings.morphism.FrobeniusEndomorphism_generic
method), 62
precompose() (sage.rings.derivation.RingDerivationWithoutTwist
method), 186
precompose() (sage.rings.derivation.RingDerivationWithTwist_generic
method), 183
primary_decomposition() (sage.rings.ideal.Ideal_generic method), 45
prime_subfield() (sage.rings.ring_field.RingField method), 13
principal_ideal() (sage.rings.ring.Ring method), 23
PrincipalIdealDomain (class in sage.rings.ring), 14
print_options() (sage.rings.ring_extension.RingExtension_generic
method), 140
prod_with_derivative() (in module sage.rings.generic), 161
ProductTree (class in sage.rings.generic), 159
pth_power() (sage.rings.derivation.RingDerivationWithoutTwist
method), 186
pushforward() (sage.rings.morphism.RingHomomorphism
method), 68

Python Enhancement Proposals
PEP 3141, 173

Q
quotientRing() (in module sage.rings.quotient_ring), 81
QuotientRing_generic (class in sage.rings.quotient_ring), 83
QuotientRing_nc (class in sage.rings.quotient_ring), 83
QuotientRingElement (class in sage.rings.quotient_ring_element), 94
QuotientRingIdeal_generic (class in sage.rings.quotient_ring), 83
QuotientRingIdeal_principal (class in sage.rings.quotient_ring), 83
RingDerivation (class in sage.rings.derivation), 178
RingDerivationModule (class in sage.rings.derivation), 178
RingDerivationWithoutTwist (class in sage.rings.derivation), 184
RingDerivationWithoutTwist_fraction_field (class in sage.rings.derivation), 187
RingDerivationWithoutTwist_function (class in sage.rings.derivation), 187
RingDerivationWithoutTwist_quotient (class in sage.rings.derivation), 188
random_element() (sage.rings.derivation.RingDerivationModule method), 181
random_element() (sage.rings.ring.RandomElement method), 104
random_element() (sage.rings.ideal.Ideal_generic method), 45
random_element() (sage.rings.ring.Ring method), 23
random_element() (sage.rings.ring_extension.RingExtension_generic method), 141
RealBallField (class in sage.rings.abc), 31
RealDoubleField (class in sage.rings.abc), 31
RealField (class in sage.rings.abc), 31
RealIntervalField (class in sage.rings.abc), 32
reduce() (sage.rings.quotient_ring_element_reduce() method), 107
reduce() (sage.rings.ring_extension.RingExtensionElement method), 108
reduce() (sage.rings.quotient_ring_element.Reduce method), 48
relative_degree() (sage.rings.ring_extension.RingExtensionGeneric method), 142
remainders() (sage.rings.generic.ProductTree method), 160
residue_field() (sage.rings.quotient_ring_element_residue_field() method), 48
retract() (sage.rings.quotient_ring.QuotientRing_nc_retract() method), 92
Ring (class in sage.rings.ring), 17
ring() (sage.rings.quotient_ring_element_ring() method), 104
ring() (sage.rings.quotient_ring_element_ring() method), 46
ring() (sage.rings.quotient_ring_element_ring() method), 52
ring() (sage.rings.quotient_ring_element_ring() method), 125
ring_of_constants() (sage.rings.quotient_ring_element_ring_of_constants() method), 181
ring_of_integers() (sage.rings.quotient_ring_element_ring_of_integers() method), 102
RingHomset (class in sage.rings.homset), 76
RingHomset_generic (class in sage.rings.homset), 76
Index

RingHomset_quo_ring (class in sage.rings.homset), 77
RingMap (class in sage.rings.morphism), 74
RingMap_lift (class in sage.rings.morphism), 75
root() (sage.rings.generic.ProductTree method), 161

S
sage.rings.abc module, 26
sage.rings.big_oh module, 163
sage.rings.derivation module, 175
sage.rings.fraction_field module, 99
sage.rings.fraction_field_element module, 105
sage.rings.generic module, 159
sage.rings.homset module, 76
sage.rings.ideal module, 35
sage.rings.ideal_monoid module, 52
sage.rings.infinity module, 165
sage.rings.localization module, 111
sage.rings.morphism module, 55
sage.rings.noncommutative_ideals module, 52
sage.rings.numbers_abc module, 173
sage.rings.quotient_ring module, 79
sage.rings.quotient_ring_element module, 94
sage.rings.ring module, 1
sage.rings.ring_extension module, 123
sage.rings.ring_extension_element module, 144
sage.rings.ring_extension_morphism module, 154
section() (sage.rings.fraction_field.FractionFieldEmbedding method), 100
side() (sage.rings.noncommutative_ideals.Ideal_nc method), 54
sign() (sage.rings.infinity.FiniteNumber method), 168
sign() (sage.rings.infinity.LessThanInfinity method), 169
SignError, 170

some_elements() (sage.rings.derivation.RingDerivationModule method), 181
some_elements() (sage.rings.fraction_field_element.FractionField_generic method), 104
specialization() (sage.rings.fraction_field_element.FractionFieldElement method), 108
sqrt() (sage.rings.infinity.FiniteNumber method), 168
sqrt() (sage.rings.infinity.MinusInfinity method), 170
sqrt() (sage.rings.infinity.PlusInfinity method), 170
sqrt() (sage.rings.ring_extension_element.RingExtensionElement method), 146
support() (sage.rings.fraction_field_element.FractionFieldElement_1poly method), 109
SymbolicRing (class in sage.rings.abc), 32

T
term_order() (sage.rings.quotient_ring.QuotientRing_nc method), 93
test_comparison() (in module sage.rings.infinity), 172
test_signed_infinity() (in module sage.rings.infinity), 173
tower_bases() (in module sage.rings.ring_extension), 143
trace() (sage.rings.ring_extension_element.RingExtensionWithBasisElement method), 152
twisting_morphism() (sage.rings.derivation.RingDerivationModule method), 182

U
underlying_map() (sage.rings.morphism.RingHomomorphism_from_base method), 71
unit_ideal() (sage.rings.ring.Ring method), 23
UniversalCyclotomicField (class in sage.rings.abc), 32
UnsignedInfinity (class in sage.rings.infinity), 170
UnsignedInfinityRing_class (class in sage.rings.infinity), 170

V
valuation() (sage.rings.fraction_field_element.FractionFieldElement method), 108
variable_names() (in module sage.rings.ring_extension), 143
variables() (sage.rings.quotient_ring_element.QuotientRingElement method), 97
vector() (sage.rings.ring_extension_element.RingExtensionWithBasisElement method), 153

Z
zero() (sage.rings.homset.RingHomset_generic method), 76
zero() (sage.rings.ring.Ring method), 23
zero_ideal() (sage.rings.ring.Ring method), 24
zeta() (sage.rings.ring.Ring method), 24
zeta_order() (sage.rings.ring.Ring method), 25