1.1 Rings

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

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

The class inheritance hierarchy is:

- \textit{Ring}
  - \textit{Algebra}
    - \textit{CommutativeRing}
      * \textit{NoetherianRing}
      * \textit{CommutativeAlgebra}
      * \textit{IntegralDomain}
        · \textit{DedekindDomain}
        · \textit{PrincipalIdealDomain}

Subclasses of \textit{PrincipalIdealDomain} are

- \textit{EuclideanDomain}
- \textit{Field}
  - \textit{FiniteField}

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

(A distinct but equally awkward issue is that sometimes we may not know \textit{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 \textit{DedekindDomain} class.)

AUTHORS:

- David Harvey (2006-10-16): changed \textit{CommutativeAlgebra} to derive from \textit{CommutativeRing} instead of from \textit{Algebra}.  

class sage.rings.ring.Algebra
Bases: sage.rings.ring.Ring

Generic algebra

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

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

EXAMPLES:

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

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

EXAMPLES:

```
sage: B = QuaternionAlgebra(2)
sage: B.has_standard_involution()
True
sage: R.<x> = PolynomialRing(QQ)
sage: K.<u> = NumberField(x**2 - 2)
sage: A = QuaternionAlgebra(K,-2,5)
sage: A.has_standard_involution()
True
sage: L.<a,b> = FreeAlgebra(QQ,2)
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: sage.rings.ring.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: sage.rings.ring.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:

- \( \text{arg} \) – (optional) a generator or a list of coefficients that defines the derivation
- \( \text{twist} \) – (optional) the twisting homomorphism

EXAMPLES:

```python
sage: R.<x,y,z> = QQ[]
sage: R.derivation()
d/dx
```

In that case, \( \text{arg} \) could be a generator:

```python
sage: R.derivation(y)
d/dy
```

or a list of coefficients:

```python
sage: R.derivation([1,2,3])
d/dx + 2*d/dy + 3*d/dz
```

It is not possible to define derivations with respect to a polynomial which is not a variable:

```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
```

(continues on next page)
specifying a scalar, the returned twisted derivation is the corresponding multiple of $\theta - id$:

\[
\text{sage: } R.derivation(1, twist=theta) \\
\{x |--> x^2, y |--> y^2, z |--> z^2\} - id
\]

\[
\text{sage: } R.derivation(x, twist=theta) \\
x*(\{x |--> x^2, y |--> y^2, z |--> z^2\} - id)
\]

```

derivation_module(codomain=None, twist=None)
Returns the module of derivations over this ring.

INPUT:

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

- twist -- a morphism from this ring to codomain or None (default: None); if None, the coercion map from this ring to codomain will be used

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

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

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: M = R.derivation_module(); M
Module of derivations over Multivariate Polynomial Ring in x, y, z over Rational Field
sage: M.gens()
(d/dx, d/dy, d/dz)
```

We can specify a different codomain:

```
sage: K = R.fraction_field()
sage: M = R.derivation_module(K); M
Module of derivations from Multivariate Polynomial Ring in x, y, z over Rational Field to Fraction Field of Multivariate Polynomial Ring in x, y, z over Rational Field
sage: M.gen() / x
1/x*d/dx
```

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

```
sage: ev = R.hom([QQ(0), QQ(1), QQ(2)])
sage: ev
Ring morphism:
  From: Multivariate Polynomial Ring in x, y, z over Rational Field
  To:   Multivariate Polynomial Ring in x, y, z over Rational Field
sage: ev
```

(continues on next page)
To: Rational Field
Defn: x |--> 0
    y |--> 1
    z |--> 2
sage: M = R.derivation_module(ev)
sage: M
Module of derivations from Multivariate Polynomial Ring in x, y, z over
    Rational Field to Rational Field

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

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

An example with a twisting homomorphism:

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 $\text{self}[x] / (f(x))$.

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 $\text{self}[x] / (f(x))$ is done anyway.

EXAMPLES:

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.<x> = PolynomialRing(GF(5))
sage: F.<a> = GF(5).extension(x^2 - 2)
sage: P.<t> = F[]
sage: R.<b> = F.extension(t^2 - a); R
Univariate Quotient Polynomial Ring in b over Finite Field in a of size 5^2
with modulus b^2 + 4*a
```

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

We can specify a power:

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

### `ideal_monoid()`

Return the monoid of ideals of this ring.

**EXAMPLES:**

```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:**
sage: QQ.is_commutative()
True
sage: ZpCA(7).is_commutative()
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()
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.

class sage.rings.ring.DedekindDomain

Bases: sage.rings.ring.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:

sage: K = NumberField(x^2 + 1, 's')
sage: OK = K.ring_of_integers()
sage: OK.integral_closure()
Gaussian Integers in Number Field in s with defining polynomial x^2 + 1
sage: OK.integral_closure() == OK
True
sage: QQ.integral_closure() == QQ
True

is_integrally_closed()
Return True since Dedekind domains are integrally closed.

EXAMPLES:

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

sage: ZZ.is_integrally_closed()
True
sage: K = NumberField(x^2 + 1, 's')
sage: OK = K.ring_of_integers()
sage: OK.is_integrally_closed()
True

These, however, are not Dedekind domains:

sage: QQ.is_integrally_closed()
True
sage: S = ZZ[sqrt(5)]; S.is_integrally_closed()

(continues on next page)
False

```python
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: K = NumberField(x^2 + 1, 's')
sage: OK = K.ring_of_integers()
sage: OK.is_noetherian()
True
sage: QQ.is_noetherian()
True
```

### krull_dimension()

Return 1 since Dedekind domains have Krull dimension 1.

**EXAMPLES:**

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

```python
sage: ZZ.krull_dimension()
1
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
sage: T.<x,y> = PolynomialRing(QQ,2); T
Multivariate Polynomial Ring in x, y over Rational Field
sage: T.krull_dimension()
2
sage: U.<x,y,z> = PolynomialRing(ZZ,3); U
Multivariate Polynomial Ring in x, y, z over Integer Ring
sage: U.krull_dimension()
4
```

```python
sage: K.<i> = QuadraticField(-1)
sage: R = K.order(2*i); R
Order in Number Field in i with defining polynomial x^2 + 1 with i = 1*I
sage: R.is_maximal()
False
sage: R.krull_dimension()
1
```
class sage.rings.ring.EuclideanDomain
Bases: sage.rings.ring.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: sage.rings.ring.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):
... Not ImplementedError: 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
```
sage: F = NumberField(x^2 + 1, 'i')
sage: F.fraction_field()
Number Field in i with defining polynomial x^2 + 1

ideal(*gens, **kwds)

Return the ideal generated by gens.

EXAMPLES:

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

integral_closure()

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

EXAMPLES:

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

is_field(proof=True)

Return True since this is a field.

EXAMPLES:

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

is_integrally_closed()

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

EXAMPLES:

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

is_noetherian()

Return True since fields are Noetherian rings.

EXAMPLES:

sage: QQ.is_noetherian()
True

krull_dimension()

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

EXAMPLES:

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

prime_subfield()

Return the prime subfield of self.

EXAMPLES:

sage: k = GF(9, 'a')
sage: k.prime_subfield()
Finite Field of size 3

class sage.rings.ring.IntegralDomain

Bases: sage.rings.ring.CommutativeRing

Generic integral domain class.

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

category instead.

is_field(proof=True)

Return True if this ring is a field.

EXAMPLES:

sage: GF(7).is_field()
True

The following examples have their own is_field implementations:

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

is_integral_domain(proof=True)

Return True, since this ring is an integral domain.

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

EXAMPLES:

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

is_integrally_closed()

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

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

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

1.1. Rings
EXAMPLES:

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

class sage.rings.ring.NoetherianRing
    Bases: sage.rings.ring.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.

```python
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: sage.rings.ring.IntegralDomain

Generic principal ideal domain.

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

```python
class_group()
    Return the trivial group, since the class group of a PID is trivial.

    EXAMPLES:

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

Note the changes to the second entry:

sage: c = (2*3)/(7*11); d = (13*17)/(7^3*19*23)
sage: factor(c); factor(d); factor(QQ.content(c,d))
2 * 3 * 7^-1 * 11^-1
7^-3 * 13 * 17 * 19^-1 * 23^-1
7^-3 * 11^-1 * 19^-1 * 23^-1

\texttt{gcd}(x, y, \texttt{coerce=True})

Return the greatest common divisor of \(x\) and \(y\), as elements of \texttt{self}.

**EXAMPLES:**

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

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

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

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

Polynomial rings over fields are GCD domains as well. Here is a simple example over the ring of polynomials over the rationals as well as over an extension ring. Note that \texttt{gcd} requires \(x\) and \(y\) to be coercible:
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^2 - 2
sage: R.base_extend(S).gcd(f, (x - a)*(x^2 - 3))
x - a

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

EXAMPLES:

sage: Zp(5).is_noetherian()  
True

class sage.rings.ring.Ring  
Bases: sage.structure.parent_gens.ParentWithGens

Generic ring class.

base_extend(R)  
EXAMPLES:

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

category()  
Return the category to which this ring belongs.

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

EXAMPLES:

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

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

```python
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:**

```python
sage: RDF.epsilon()
2.220446049250313e-16
sage: ComplexField(53).epsilon()
2.22044604925031e-16
sage: RealField(10).epsilon()
0.0020
```

For exact rings, zero is returned:

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

This also works over derived rings:

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

For the symbolic ring, there is no reasonable answer:

```python
sage: SR.epsilon()
Traceback (most recent call last):
... 
NotImplementedError
```

**ideal(***)

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,x^3+y^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)
sage: A.ideal(A.1,A.2^2)
Twosided Ideal (Sq(2), Sq(2,2)) of mod 2 Steenrod algebra, milnor basis
sage: A.ideal(A.1,A.2^2,side='left')
Left Ideal (Sq(2), Sq(2,2)) of mod 2 Steenrod algebra, milnor basis

ideal_monoid()

Return the monoid of ideals of this ring.

EXAMPLES:

sage: F.<x,y,z> = FreeAlgebra(ZZ, 3)
sage: I = F*[x*y+y*z,x^2+x*y-y*x-y^2]*F
sage: Q = sage.rings.ring.Ring.quotient(F,I)
sage: Q.ideal_monoid()
Monoid of ideals of Quotient of Free Algebra on 3 generators (x, y, z) over Integer Ring by the ideal (x*y + y*z, x^2 + x*y - y*x - y^2)

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

is_commutative()

Return True if this ring is commutative.

EXAMPLES:

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

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

EXAMPLES:
\begin{verbatim}
sage: QQ.is_exact()  # indirect doctest
True
sage: ZZ.is_exact()
True
sage: Qp(7).is_exact()
False
sage: Zp(7, type='capped-abs').is_exact()
False
\end{verbatim}

**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:**
\begin{verbatim}
sage: QQ.is_field()
True
sage: GF(9,'a').is_field()
True
sage: ZZ.is_field()
False
sage: QQ['x'].is_field()
False
sage: Frac(QQ['x']).is_field()
True
\end{verbatim}

This illustrates the use of the proof parameter:
\begin{verbatim}
sage: R.<a,b> = QQ[]
sage: S.<x,y> = R.quo((b^3))
sage: S.is_field(proof = True)
Traceback (most recent call last):
... NotImplementedError
sage: S.is_field(proof = False)
False
\end{verbatim}

**is_integral_domain** (proof=True)
Return True if this ring is an integral domain.

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

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

**EXAMPLES:**
sage: QQ.is_integral_domain()
True
sage: ZZ.is_integral_domain()
True
sage: ZZ['x,y,z'].is_integral_domain()
True
sage: Integers(8).is_integral_domain()
False
sage: Zp(7).is_integral_domain()
True
sage: Qp(7).is_integral_domain()
True
sage: R.<a,b> = ZZ[]

This illustrates the use of the proof parameter:

sage: S.<x,y> = R.quo((b^3))
sage: S.is_integral_domain(proof = True)
Traceback (most recent call last):
  ... NotImplmentedError
sage: S.is_integral_domain(proof = False)
False

is_noetherian()
Return True if this ring is Noetherian.

EXAMPLES:

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

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

EXAMPLES:

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

is_ring()
Return True since self is a ring.
EXAMPLES:

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

**is_subring**(other)

Return True if the canonical map from self to other is injective.

Raises a Not Implemented Error if not known.

EXAMPLES:

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

**one()**

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

EXAMPLES:

```
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()
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
```

**quo**(I, names=None)

Create the quotient of R by the ideal I. This is a synonym for **quotient()**

EXAMPLES:
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = R.quo((x^2, y))
sage: S
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the → ideal (x^2, y)
sage: S.gens()
(a, 0)
sage: a == b
False

quotient (I, names=None)
Create the quotient of this ring by a twosided ideal I.

INPUT:

• I – a twosided ideal of this ring, R.
• names – (optional) names of the generators of the quotient (if there are multiple generators, you can specify a single character string and the generators are named in sequence starting with 0).

EXAMPLES:

sage: R.<x> = PolynomialRing(ZZ)
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: S = R.quotient(I, 'a')
sage: S.gens()
(a,)
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = R.quotient((x^2, y))
sage: S
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the → ideal (x^2, y)
sage: S.gens()
(a, 0)
sage: a == b
False

quotient_ring (I, names=None)
Return the quotient of self by the ideal I of self. (Synonym for self.quotient(I).)

INPUT:

• I – an ideal of R
• names – (optional) names of the generators of the quotient. (If there are multiple generators, you can specify a single character string and the generators are named in sequence starting with 0.)

OUTPUT:

• R/I – the quotient ring of 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, 'a')
sage: S.gens()
(a,)

(continues on next page)
\begin{verbatim}
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = R.quotient_ring((x^2, y))
sage: S
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the →ideal (x^2, y)
sage: S.gens()
(a, 0)
sage: a == b
False

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:

\begin{verbatim}
sage: Zp(7).unit_ideal()
Principal ideal (1 + O(7^20)) of 7-adic Ring with capped relative precision 20
\end{verbatim}

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

EXAMPLES:

\begin{verbatim}
sage: ZZ.zero()
0
sage: QQ.zero()
0
sage: QQ['x'].zero()
0
The result is cached:

\begin{verbatim}
sage: ZZ.zero() is ZZ.zero()
True
\end{verbatim}

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

EXAMPLES:

\begin{verbatim}
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:
\end{verbatim}
\end{verbatim}

1.1. Rings

```
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)
zeta6
sage: CyclotomicField(3).zeta(3)
zeta3
sage: CyclotomicField(3).zeta(3).multiplicative_order()
3
sage: a = GF(7).zeta(); a
3
sage: a.multiplicative_order()
6
sage: a = GF(49,'z').zeta(); a
z
sage: a.multiplicative_order()
48
sage: a = GF(49,'z').zeta(2); a
6
sage: a.multiplicative_order()
2
sage: QQ.zeta(3)
Traceback (most recent call last):
  ...
ValueError: no n-th root of unity in rational field
```

**zeta_order()**

Return the order of the distinguished root of unity in \(self\).

**EXAMPLES:**

```
sage: CyclotomicField(19).zeta_order()
38
sage: GF(19).zeta_order()
18
sage: GF(5^3,'a').zeta_order()
124
```

(continues on next page)
sage: Zp(7, prec=8).zeta_order()
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)
sage: is_Ring(MS)
True
2.1 Ideals of commutative rings

Sage provides functionality for computing with ideals. One can create an ideal in any commutative or non-commutative ring $R$ by giving a list of generators, using the notation $R.\text{ideal}([a,b,...])$. The case of non-commutative rings is implemented in $\text{noncommutative_ideals}$. 

A more convenient notation may be $R*[a,b,...]$ or $[a,b,...]*R$. If $R$ is non-commutative, the former creates a left and the latter a right ideal, and $R*[a,b,...]*R$ creates a two-sided ideal.

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

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

 Note:  R will be set as the active ring in Singular

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

 sage: P.<x,y,z> = PolynomialRing(QQ,3,order='lex')
sage: I = sage.rings.ideal.Cyclic(P)
sage: 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]

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

 sage: R.<x,y,z,t,u,v> = QQ['x,y,z,t,u,v']
sage: I = sage.rings.ideal.Cyclic(R,6)
sage: B = I.groebner_basis()
sage: len(B)
45
\end{verbatim}
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)
sage: I = sage.rings.ideal.FieldIdeal(P); I
Ideal (x^2 + x, y^2 + y) of Multivariate Polynomial Ring in x, y over Finite Field of size 2
```

Another, similar example:

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

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

Create the ideal in ring with given generators.

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

- R.ideal(gens, coerce=True)
- gens*R
- R*gens

**INPUT:**

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

**OUTPUT:** The ideal of ring generated by `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
```
This example illustrates how Sage finds a common ambient ring for the ideal, even though 1 is in the integers (in this case).

This shows that the issues at trac ticket #1104 are resolved:

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

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

You can also pass in a specific ideal type:

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

2.1. Ideals of commutative rings

27
Fractional ideal of a ring.

See `Ideal()`.

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

Bases: `sage.structure.element.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:**

```
sage: R.<t> = GF(9, names='a')[
```
```
sage: I = R.ideal(t^4 + t + 1)
```
```
sage: I.absolute_norm()
```
```
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:**
base_ring()

Returns the base ring of this ideal.

EXAMPLES:

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

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

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

And $p$-adic numbers:

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

category()

Return the category of this ideal.

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

EXAMPLES:

```
sage: 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
ever Integer Ring
sage: K.category()
Category of ring ideals in Univariate Polynomial Ring in x
ever 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()
[Ideal (y, x) of Multivariate Polynomial Ring in x, y over Rational Field]
```

**gen(i)**

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

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

**gens_reduced()**

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

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

**EXAMPLES:**

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

**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:**
is_maximal()  
Returns True if this ideal is maximal.

is_primary(P=None)  
Returns True if this ideal is primary (or $P$-primary, if a prime ideal $P$ is specified).

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

INPUT:

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

EXAMPLES:

Some examples from the Macaulay2 documentation:

is_prime()  
Return True if this ideal is prime.

EXAMPLES:
The code below demonstrates the use of the `is_prime()` method for ideals in various rings.

```python
sage: R.<x, y> = QQ[]
sage: I = R.ideal([x, y])
sage: I.is_prime()       # a maximal ideal
True
sage: I = R.ideal([x^2-y])
sage: I.is_prime()       # a non-maximal prime ideal
True
sage: I = R.ideal([x^2, y])
sage: I.is_prime()       # a non-prime primary ideal
False
sage: I = R.ideal([x^2, x*y])
sage: I.is_prime()       # a non-prime non-primary ideal
False
sage: S = Integers(8)
sage: S.ideal(0).is_prime()  # no prime ideal
False
sage: S.ideal(2).is_prime()  # a prime ideal
True
sage: S.ideal(4).is_prime()  # not prime ideal
False
```

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

```python
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:**

```python
sage: R = ZZ['x']
sage: I = R.ideal(2,x)
sage: I.is_principal()  # should be False
Traceback (most recent call last):
  ...                               
NotImplementedError
sage: J = R.base_extend(QQ).ideal(2,x)
sage: J.is_principal()  # should be True
True
```

### is_trivial()

Return `True` if this ideal is `(0)` or `(1)`.
minimal_associated_primes()
Return the list of minimal associated prime ideals of this ideal.

EXAMPLES:

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

norm()
Returns the norm of this ideal.

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

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

EXAMPLES:

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

primary_decomposition()
Return a decomposition of this ideal into primary ideals.

EXAMPLES:

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

random_element(*args, **kwds)
Return a random element in this ideal.

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

**reduce** (*f*)

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

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

**EXAMPLES:**

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

**ring** (*f*)

Returns the ring containing this ideal.

**EXAMPLES:**

```python
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 self.ring() is different from self.base_ring()  

```python
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:

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

```python
sage: Rbar = R.quotient(I, names='a')  
sage: S = PolynomialRing(Rbar, 'y'); y = Rbar.gen(); S  
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)  
sage: J.ring()  
Univariate Polynomial Ring in y over Univariate Quotient Polynomial Ring in a  
˓→over Rational Field with modulus x^2 - 3
```

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

**Bases:** `sage.rings.ideal.Ideal_principal`

An ideal of a principal ideal domain.
gcd(other)
Returns the greatest common divisor of the principal ideal with the ideal other; that is, the largest principal ideal contained in both the ideal and other

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

Note: The second example is broken.

EXAMPLES:
An example in the principal ideal domain \( \mathbb{Z} \):

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

is_maximal()
Returns whether this ideal is maximal.

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

EXAMPLES:

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

is_prime()
Returns True if the ideal is prime.

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

EXAMPLES:

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

(continues on next page)
In fields, only the zero ideal is prime:

```sage
sage: RR.ideal(0).is_prime()
True
sage: RR.ideal(7).is_prime()
False
```

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

**EXAMPLES:**

```sage
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
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
```

(continues on next page)
To: Integer Ring
sage: lift(F(12345/67890))
33
sage: (12345/67890) % 61
33

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

gen()
   Returns the generator of the principal ideal. The generators are elements of the ring containing the ideal.

EXAMPLES:

A simple example in the integers:

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

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

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

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

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

```python
sage: R.<x> = ZZ[]
sage: I = R.ideal(x)
sage: J = R.ideal(2,x)
sage: K = R.base_extend(QQ).ideal(2,x)
sage: I
Principal ideal (x) of Univariate Polynomial Ring in x over Integer Ring
sage: J
Ideal (2, x) of Univariate Polynomial Ring in x over Integer Ring
sage: K
Principal ideal (1) of Univariate Polynomial Ring in x over Rational Field
sage: I.is_principal()
True
sage: K.is_principal()
True
```

`sage.rings.ideal.Katsura(R, n=None, homog=False, singular=Singular)`

\(n\)-th katsura ideal of \(R\) if \(R\) is coercible to Singular.

**INPUT:**

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

**EXAMPLES:**

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

`sage.rings.ideal.is_Ideal(x)`

Return True if object is an ideal of a ring.

**EXAMPLES:**

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

```python
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:

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

```sage
is_Ideal(I)
True
```

```sage
is_Ideal((x^2 + 1)*R)
True
```

### 2.2 Monoid of ideals in a commutative ring

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

```sage
MS = MatrixSpace(QQ,3,3)
type(MS.ideal(MS.one()).parent())
<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: rings.ideal_monoid.IdealMonoid(R)
```

```
class sage.rings.ideal_monoid.IdealMonoid_c(R)
Bases: sage.structure.parent.Parent
```

```
IdealMonoid of ideals of Univariate Polynomial Ring in x over Rational Field
```

```
class sage.rings.ideal_monoid.IdealMonoid_c(R)
Bases: sage.structure.parent.Parent
```

```
The monoid of ideals in a commutative ring.
```

**Element**

```
alias of sage.rings.ideal.Ideal_generic
ring()
```

```
Return the ring of which this is the ideal monoid.
```

**EXAMPLES:**

```sage
R = QuadraticField(-23, 'a')
sage: rings.ideal_monoid.IdealMonoid(R).ring()
```

### 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>`, trac ticket #7797.

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

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

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

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

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

EXAMPLES:

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

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

Generic non-commutative ideal.

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

EXAMPLES:

sage: MS = MatrixSpace(QQ,2,2)
sage: I = MS*[MS.1,MS.2]; I
Left Ideal
([ 0 1]
 [0 0],

(continues on next page)
of Full MatrixSpace of 2 by 2 dense matrices over Rational Field

```
sage: MS*[MS.1, MS.2]*MS
           Right Ideal
        (0 1)
        (0 0),

        (0 0)
        (1 0)

        of Full MatrixSpace of 2 by 2 dense matrices over Rational Field

sage: MS*[MS.1, MS.2]*MS
           Twosided Ideal
        (0 1)
        (0 0),

        (0 0)
        (1 0)

        of Full MatrixSpace of 2 by 2 dense matrices over Rational Field
```

side()

Return a string that describes the sidedness of this ideal.

EXAMPLES:

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

There is no homomorphism in the other direction:

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

EXAMPLES:

Reduction to finite field:

```sage
sage: H = Hom(ZZ, GF(9, 'a'))
sage: phi = H([1])
sage: phi(5)
2
sage: psi = H([4])
sage: psi(5)
2
```

Map from single variable polynomial ring:
\begin{lstlisting}[language=sage]
sage: R.<x> = ZZ[]
sage: phi = R.hom([2], GF(5))
sage: phi
Ring morphism:
  From: Univariate Polynomial Ring in x over Integer Ring
  To:   Finite Field of size 5
  Defn: x |--> 2
sage: phi(x + 12)
4
Identity map on the real numbers:
sage: f = RR.hom([1]); f
Ring endomorphism of Real Field with 53 bits of precision
  Defn: 1.00000000000000 |--> 1.00000000000000
sage: f(2.5)
2.50000000000000
sage: f = RR.hom([2.0])
Traceback (most recent call last):
  ... ValueError: relations do not all (canonically) map to 0 under map determined by images of generators

Homomorphism from one precision of field to another.
From smaller to bigger doesn’t make sense:
sage: R200 = RealField(200)
sage: f = RR.hom( R200 )
Traceback (most recent call last):
  ... TypeError: natural coercion morphism from Real Field with 53 bits of precision to Real Field with 200 bits of precision not defined

From bigger to small does:
sage: f = RR.hom( RealField(15) )
sage: f(2.5)
2.500
sage: f(RR.pi())
3.142

Inclusion map from the reals to the complexes:
\begin{lstlisting}[language=sage]
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
\end{lstlisting}

A map from a multivariate polynomial ring to itself:
\begin{lstlisting}[language=sage]
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
\end{lstlisting}
\end{lstlisting}
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
  Defn: Choice of lifting map
sage: pi.lift(13)
5
```

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

```
sage: k = GF(2)
sage: i = k.hom(GF(4, 'a'))
sage: i
Ring morphism:
  From: Finite Field of size 2
  To:   Finite Field in a of size 2^2
  Defn: 1 |--> 1
sage: i(0)
0
sage: a = i(1); a.parent()
Finite Field in a of size 2^2
```
We next compose the inclusion with reduction from the integers to \( \mathbb{F}_2 \):

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

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

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

An automorphism of a quotient of a univariate polynomial ring:

```python
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:

```python
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); 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: R.set_default_prec(10)
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))
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 + 3*t^5 + 3*t^10 + t^15 + O(t^20)
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
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^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
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:

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

Embedding a number field into the reals:

```python
sage: R.<x> = PolynomialRing(QQ)
sage: K.<beta> = NumberField(x^3 - 2)
sage: alpha = RR(2)^(1/3); alpha
1.25992104989487
sage: i = K.hom([alpha],check=False); i
Ring morphism:
   From: Number Field in beta with defining polynomial x^3 - 2
   To: Real Field with 53 bits of precision
   Defn: beta |--> 1.25992104989487
sage: i(beta)
1.25992104989487
sage: i(beta^3)
2.00000000000000
sage: i(beta^2 + 1)
2.58740105196820
```

An example from Jim Carlson:

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

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

```
sage: K.<u> = PowerSeriesRing(GF(5))
sage: Frob = K.frobenius_endomorphism()
sage: Frob.power()
1
sage: (Frob^9).power()
9
```

**class sage.rings.morphism.RingHomomorphism**

Bases: `sage.rings.morphism.RingMap`

Homomorphism of rings.

**inverse_image(I)**

Return the inverse image of the ideal \( I \) under this ring homomorphism.

**EXAMPLES:**

This is not implemented in any generality yet:

```
sage: f = ZZ.hom(Zp(2))
sage: f.inverse_image(ZZ.ideal(2))
Traceback (most recent call last):
  ... Not Implemented Error
```

**lift(x=None)**

Return a lifting homomorphism 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()
sage: f.pushforward(R.ideal([x,3*x+x*y+y^2]))
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: sage.rings.morphism.RingHomomorphism

A ring homomorphism that is a coercion.

class sage.rings.morphism.RingHomomorphism_cover
     Bases: sage.rings.morphism.RingHomomorphism

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

EXAMPLES:

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

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

class sage.rings.morphism.RingHomomorphism_from_base
     Bases: sage.rings.morphism.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:

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:

sage: PR.<t> = R[]
sage: PS = S['t']
sage: Pf = PR.hom(f,PS)
sage: Pf

(continues on next page)
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

\[
\begin{align*}
\text{Ring morphism:} \\
\text{From: Multivariate Polynomial Ring in } x, y \text{ over Rational Field} \\
\text{To: Univariate Polynomial Ring in } z \text{ over Rational Field} \\
\text{Defn: } x \mapsto 2z \\
y \mapsto 3z
\end{align*}
\]

\[
sage: p = (x - 4*y + 1/13)*t^2 + (1/2*x^2 - 1/3*y^2)*t + 2*y^2 + x
\]
\[
sage: Pf(p)
\]
\[
(-10*z + 1/13)*t^2 - z^2*t + 18*z^2 + 2*z
\]

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

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

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

\[
sage: MPR = \text{MatrixSpace}(PR, 2)
\]
\[
sage: MPS = \text{MatrixSpace}(PS, 2)
\]
\[
sage: M = MPR(\begin{bmatrix} (-x + y)*t^2 + 58*t - 3*x^2 + x*y, & (-1/7*x*y - 1/40*x)*t^2 + (5*x^2 + y^2)*t + 2*y, & (-1/3*y + 1)*t^2 + 1/3*x*y + y^2 + 5/2*y + 1/4, & (x + 6*y + 1)*t^2 \end{bmatrix})
\]
\[
sage: MPf = MPR.hom(f, MPS); MPf
\]
\[
\text{Ring morphism:}
\begin{align*}
\text{From: Full MatrixSpace of 2 by 2 dense matrices over Univariate Polynomial Ring} \\
in t over Multivariate Polynomial Ring in x, y over Rational Field \\
\text{To: Full MatrixSpace of 2 by 2 dense matrices over Univariate Polynomial Ring} \\
in t over Univariate Polynomial Ring in z over Rational Field \\
\text{Defn: Induced from base ring by}
\end{align*}
\]
\[
\text{Ring morphism:}
\begin{align*}
\text{From: Univariate Polynomial Ring in } t \text{ over Multivariate Polynomial Ring} \\
in x, y over Rational Field \\
\text{To: Univariate Polynomial Ring in } t \text{ over Univariate Polynomial Ring} \\
in z over Rational Field \\
\end{align*}
\]
Defn: Induced from base ring by
Ring morphism:
  From: Multivariate Polynomial Ring in x, y over Rational Field
  To:  Univariate Polynomial Ring in z over Rational Field
  Defn: x |--> 2*z
        y |--> 3*z

sage: MPf(M)
[                    z*t^2 + 58*t - 6*z^2 (-6/7*z^2 - 1/20*z)*t^2 + 29*z^2*t +
  \downarrow \downarrow 6*z]
[ (-z + 1)*t^2 + 11*z^2 + 15/2*z + 1/4 (20 *z + 1)*t^2 + \downarrow \downarrow 2]

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

EXAMPLES:

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

class sage.rings.morphism.RingHomomorphism_from_quotient
Bases: sage.rings.morphism.RingHomomorphism

A ring homomorphism with domain a generic quotient ring.

INPUT:
  * parent -- a ring homset \text{Hom}(R,S)
  * phi -- a ring homomorphism \text{C} \rightarrow \text{S}, where \text{C} is the domain of \text{R.cover()}

OUTPUT: a ring homomorphism

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

EXAMPLES:

sage: R.<x, y, z> = PolynomialRing(QQ, 3)
sage: S.<a, b, c> = R.quo(x^3 + y^3 + z^3)
sage: phi = S.hom([b, c, a]); phi
Ring endomorphism of Quotient of Multivariate Polynomial Ring in x, y, z over Rational Field by the ideal (x^3 + y^3 + z^3)
  Defn: a |--> b
        b |--> c
        c |--> a
sage: phi(a+b+c)
a + b + c
sage: loads(dumps(phi)) == phi
True

Validity of the homomorphism is determined, when possible, and a \text{TypeError} is raised if there is no homomorphism sending the generators to the given images.
sage: S.hom([b^2, c^2, a^2])
Traceback (most recent call last):
...  
TypeError: images do not define a valid homomorphism

**morphism_from_cover()**

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

**EXAMPLES:**

```python
sage: R.<x,y> = QQ[]; S.<xx,yy> = R.quo([x^2,y^2])
sage: S.hom([yy,xx]).morphism_from_cover()
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:** sage.rings.morphism.RingHomomorphism

A ring homomorphism determined by the images of generators.

**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:**

```python
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:

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

### Class sage.rings.morphism.RingMap

**Bases:** sage.categories.morphism.Morphism

Set-theoretic map between rings.

### Class sage.rings.morphism.RingMap_lift

**Bases:** sage.rings.morphism.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 $R \rightarrow S$ sending $x$ to $x.lift()$.

**EXAMPLES:**

```python
sage: R.<x,y> = QQ[]
sage: S.<xbar,ybar> = R.quo( (x^2 + y^2, y) )
sage: S.lift()
```

(continues on next page)
Set-theoretic ring morphism:
- From: Quotient of Multivariate Polynomial Ring in x, y over Rational Field by
  \(\langle x^2 + y^2, y \rangle\)
- To: Multivariate Polynomial Ring in x, y over Rational Field
- Defn: Choice of lifting map
  
  ```sage```
  S.lift() == 0
  False

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

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

```sage```
return sage.rings.morphism.is_RingHomomorphism(phi)
Return True if phi is of type `RingHomomorphism`.

**EXAMPLES:**

```sage```
f = Zmod(8).cover()
sage: sage.rings.morphism.is_RingHomomorphism(f)
doctest:warning...
DeprecationWarning: is_RingHomomorphism() should not be used anymore. Check whether the category_for() your morphism is a subcategory of Rings() instead. See http://trac.sagemath.org/23204 for details.
True
```

```sage```
sage: sage.rings.morphism.is_RingHomomorphism(2/3)
False
```

### 3.2 Space of homomorphisms between two rings

```sage```
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
```

```sage```
class sage.rings.homset.RingHomset_generic(R, S, category=None)
Bases: sage.categories.homset.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

has_coerce_map_from(x)
The default for coercion maps between ring homomorphism spaces is very restrictive (until more imple-
mentation 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: Hom(ZZ, Zmod(1)).zero()
Ring morphism:
    From: Integer Ring
    To:   Ring of integers modulo 1
    Defn: 1 |--> 0
sage: Hom(ZZ, Zmod(2)).zero()
Traceback (most recent call last):
  ... ValueError: homset has no zero element

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

Bases: sage.rings.homset.RingHomset_generic

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

EXAMPLES:

sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = R.quotient(x^2 + y^2)
sage: phi = S.hom([b,a]); phi
Ring endomorphism of Quotient of Multivariate Polynomial Ring in x, y over \( \mathbb{Q} \)
\( \mapsto \) Rational Field by the ideal \((x^2 + y^2)\)

Defn: \( a \mapsto b \)

\( \begin{array}{c}
\text{sage: } \phi(a) \\
\text{sage: } \phi(b) \\
\end{array} \)

\[
\text{sage.rings.homset.} \text{is\_RingHomset}(H) \\
\text{Return True if } H \text{ is a space of homomorphisms between two rings.}
\]

\textbf{EXAMPLES:}

\[
\text{sage: from sage.rings.homset import is\_RingHomset as is\_RH} \\
\text{sage: is\_RH(Hom(ZZ, QQ))} \\
\text{True} \\
\text{sage: is\_RH(ZZ)} \\
\text{False} \\
\text{sage: is\_RH(Hom(RR, CC))} \\
\text{True} \\
\text{sage: is\_RH(Hom(FreeModule(ZZ,1), FreeModule(QQ,1)))} \\
\text{False}
\]
CHAPTER
FOUR

QUOTIENT RINGS

4.1 Quotient Rings

AUTHORS:

• William Stein

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

• Simon King (2011-04): Quotients of non-commutative rings by twosided ideals.

Todo: The following skipped tests should be removed once trac ticket #13999 is fixed:

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

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

```
sage: from sage.rings.noncommutative_ideals import Ideal_nc
sage: from itertools import product
sage: class PowerIdeal(Ideal_nc):
....:     def __init__(self, R, n):
....:         self._power = n
....:     def _reduce(self, x):
....:         R = self.ring()
....:         return add([c*R(m) for m,c in x if len(m)<self._power],R(0))

sage: F.<x,y,z> = FreeAlgebra(QQ, 3)
sage: I3 = PowerIdeal(F,3); I3
Twosided Ideal (x^3, x^2*y, x^2*z, x*y*x, x*y^2, x*y*z, x*z*x, x*z*y, x*z^2, y*x^2, y*x*y, y*x*z, y^2*x, 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^2, z*y*z, z^2*x, z^2*y, z^2*z) of Free Algebra on 3 generators (x, y, z) over Rational Field
```

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

sage: Q3.is_commutative()
False

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

sage: a*(b+c)-(b+c)*a==F.zero()
True

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

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

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

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

sage.rings.quotient_ring.QuotientRing(R, I, names=None)
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 $R/I$. 
OUTPUT: $R/I$ - the quotient ring $R$ mod the ideal $I$

ASSUMPTION:

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

EXAMPLES:

Some simple quotient rings with the integers:

\[
\begin{align*}
\text{sage: } & R = \text{QuotientRing}(\mathbb{Z}, 7\mathbb{Z}); R \\
& \text{Quotient of Integer Ring by the ideal (7)} \\
\text{sage: } & R.gens() \\
& (1,) \\
\text{sage: } & 1*R(3); 6*R(3); 7*R(3) \\
& 3 \\
& 4 \\
& 0 \\
\text{sage: } & S = \text{QuotientRing}(\mathbb{Z}, 8\mathbb{Z}); S \\
& \text{Quotient of Integer Ring by the ideal (8)} \\
\text{sage: } & 2*S(4) \\
& 0
\end{align*}
\]

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

\[
\begin{align*}
\text{sage: } & P.<x> = \mathbb{Q}[x] \\
\text{sage: } & R.\langle xx \rangle = \text{QuotientRing}(P, P.\text{ideal}(x^2 + 1)) \\
\text{sage: } & R \\
& \text{Univariate Quotient Polynomial Ring in xx over Rational Field with modulus } x^2 + 1 \\
\text{sage: } & R.gens(); R.\text{gen()} \\
& (xx,;) \\
& xx \\
\text{sage: } & \textbf{for } n \text{ in range(4): } xx^n \\
& 1 \\
& xx \\
& -1 \\
& -xx \\
\text{sage: } & P.<x> = \mathbb{Q}[x] \\
\text{sage: } & S = \text{QuotientRing}(P, P.\text{ideal}(x^2 - 2)) \\
\text{sage: } & S \\
& \text{Univariate Quotient Polynomial Ring in xbar over Rational Field with modulus } x^2 - 2 \\
\text{sage: } & xbar = S.\text{gen(); S.}\text{gen()} \\
& xbar \\
\text{sage: } & \textbf{for } n \text{ in range(3): } xbar^n \\
& 1 \\
& xbar \\
& 2
\end{align*}
\]

Sage coerces objects into ideals when possible:

\[
\begin{align*}
\text{sage: } & P.<x> = \mathbb{Q}[x] \\
\text{sage: } & R = \text{QuotientRing}(P, x^2 + 1); R \\
& \text{Univariate Quotient Polynomial Ring in xbar over Rational Field with modulus } x^2 + 1
\end{align*}
\]
By Noether’s homomorphism theorems, the quotient of a quotient ring of \( R \) is just the quotient of \( R \) by the sum of the ideals. In this example, we end up modding out the ideal \((x)\) from the ring \( \mathbb{Q}[x, y] \):

```
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = QuotientRing(R, R.ideal(1 + y^2))
sage: T.<c,d> = QuotientRing(S, S.ideal(a))
sage: T
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal \((x, y^2 + 1)\)
sage: R.gens(); S.gens(); T.gens()
(x, y) (a, b) (0, d)
sage: for n in range(4): d^n
1 d -1 -d
```

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

Bases: sage.rings.quotient_ring.QuotientRing_nc, sage.rings.ring.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=None, category=None)

Bases: sage.rings.ring.Ring, sage.structure.parent_gens.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')
sage: I = F*[x*y+y*z,x^2+x*y-y*x-y^2]*F
sage: Q.<a,b,c> = F.quo(I); Q
Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) over\( \mathbb{Q} \) by the ideal (x*y + y*z, x*x + x*y - y*x - y*y)
sage: a*b
-b*c
t sage: a^3
-b*c*a - b*c*b - b*c*c
```

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

```
sage: J = Q*[a^3-b^3]*Q
sage: R.<i,j,k> = Q.quo(J); R
Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) over \( \mathbb{Q} \) by the ideal (-y*y*z - y*z*x - 2*y*z^2, x*y + y*z, x*x + x*y - y*x)
(continues on next page)
```
sage: i^3
-j*k*i - j*k*j - j*k*k
sage: j^3
-j*k*i - j*k*j - j*k*k

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

EXAMPLES:

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

```python
sage: R.<x,y> = PolynomialRing(QQ)
sage: S.<a,b> = R.quo(x^2 + y^2)
sage: a^2 + b^2 == 0
True
sage: S(0) == a^2 + b^2
True
```

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

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

```python
sage: T.gens()
(0, d)
```

**Element**

alias of `sage.rings.quotient_ring_element.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.cover_ring()
Integer Ring
```

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

**characteristic()**

Return the characteristic of the quotient ring.
Todo: Not yet implemented!

EXAMPLES:

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

`construction()`

Returns the functorial construction of `self`.

EXAMPLES:

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

`cover()`

The covering ring homomorphism $R \rightarrow R/I$, equipped with a section.

EXAMPLES:

```sage
sage: R = ZZ.quo(3*ZZ)
sage: pi = R.cover()
sage: pi
Ring morphism:
  From: Integer Ring
  To:   Ring of integers modulo 3
  Defn: Natural quotient map
sage: pi(5)
2
sage: l = pi.lift()
sage: R.<x,y> = PolynomialRing(QQ)
sage: Q = R.quo( (x^2, y^2) )
sage: pi = Q.cover()
sage: pi(x^3+y)
ybar
sage: l = pi.lift(x+y^3)
sage: l
x
sage: l = pi.lift(); l
Set-theoretic ring morphism:
  From: Quotient of Multivariate Polynomial Ring in x, y over Rational Field
  by the ideal (x^2, y^2)
  To:   Multivariate Polynomial Ring in x, y over Rational Field
  Defn: Choice of lifting map
```

(continues on next page)
cover_ring()
Returns the cover ring of the quotient ring: that is, the original ring $R$ from which we modded out an ideal, $I$.

EXAMPLES:

```
sage: Q = QuotientRing(ZZ, 7*ZZ)
sage: Q.cover_ring()
Integer Ring
```
```
sage: P.<x> = QQ[]
sage: Q = QuotientRing(P, x^2 + 1)
sage: Q.cover_ring()
Univariate Polynomial Ring in x over Rational Field
```

defining_ideal()
Returns the ideal generating this quotient ring.

EXAMPLES:

In the integers:

```
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 is actually a quotient by a sum of two ideals:

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

```
sage: R = QuotientRing(ZZ, 7*ZZ)
sage: R.gen(0)
1
```
```
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = QuotientRing(R, R.ideal(1 + y^2))
sage: T.<c,d> = QuotientRing(S, S.ideal(a))
sage: T
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the \rightarrow ideal (x, y^2 + 1)
sage: R.gen(0); R.gen(1)
x
```

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

Return the ideal of self with the given generators.

EXAMPLES:

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

is_commutative()

Tell whether this quotient ring is commutative.

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

**AUTHOR:**

- Simon King (2011-03-23): See trac ticket #7797.

**EXAMPLES:**

Any quotient of a commutative ring is commutative:

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

The non-commutative case is more interesting:

```
sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
sage: I = F*[x*y+y*z,x^2+x*y-y*x-y^2]*F
sage: Q = F.quo(I)
sage: Q.is_commutative()
False
sage: Q.1*Q.2==Q.2*Q.1
False
```

In the next example, the generators apparently commute:

```
sage: J = F*[x*y-y*x,x*z-z*x,y*z-z*y,x^3-y^3]*F
sage: R = F.quo(J)
sage: R.is_commutative()
True
```
**is_field**(\(proof=True\))

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

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

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

**EXAMPLES:**

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

**is_noetherian()**

Return True if this ring is Noetherian.

**EXAMPLES:**

```python
sage: R = QuotientRing(ZZ, 102*ZZ)
sage: R.is_noetherian()
True
sage: P.<x> = QQ[]
sage: R = QuotientRing(P, x^2+1)
sage: R.is_noetherian()
True
```

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

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

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

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

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

**EXAMPLES:**

```
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S = R.quotient(x^2 + y^2)
sage: S.lift()
Set-theoretic ring morphism:
  From: Quotient of Multivariate Polynomial Ring in x, y over Rational Field
       by the ideal (x^2 + y^2)
  To:  Multivariate Polynomial Ring in x, y over Rational Field
       Defn: Choice of lifting map
sage: S.lift(S.0) == x
True
```

```
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
Ring morphism:
  From: Multivariate Polynomial Ring in x, y over Rational Field
  To:  Quotient of Multivariate Polynomial Ring in x, y over Rational Field
       by the ideal (x^2 + y^2)
  Defn: Natural quotient map
sage: L = S.lifting_map(); L
Set-theoretic ring morphism:
  From: Quotient of Multivariate Polynomial Ring in x, y over Rational Field
       by the ideal (x^2 + y^2)
  To:  Multivariate Polynomial Ring in x, y over Rational Field
       Defn: Choice of lifting map
sage: L(S.0)
x
sage: L(S.1)
y
```

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

```
sage: z = pi(x^3 + 2*y^2); z
-xbar*ybar^2 + 2*ybar^2
sage: L(z)
-x*y^2 + 2*y^2
sage: L(z) == x^3 + 2*y^2
False
```

Test that there also is a lift for rings that are no instances of \texttt{Ring} (see trac ticket \#11068):

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

(continues on next page)
To: Full MatrixSpace of 2 by 2 dense matrices over Finite Field of size 5
Defn: Choice of lifting map

ngens ()

Returns the number of generators for this quotient ring.

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

EXAMPLES:

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

S.<a,b> = QuotientRing(R, R.ideal(1 + y^2))
sage: S.gens(); S.ngens()
((a, b), 2)

T.<c,d> = QuotientRing(S, S.ideal(a))
sage: T.gens(); T.ngens()
((0, d), 2)
```

retract (x)

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

INPUT:

* x – An element of the cover ring

OUTPUT:

The image of the given element in self.

EXAMPLES:

```sage
R.<x,y> = PolynomialRing(QQ, 2)
sage: S = R.quotient(x^2 + y^2)
sage: S.retract((x+y)^2)
2*xbar*ybar
```
term_order()  
Return the term order of this ring.

EXAMPLES:

```
sage: P.<a,b,c> = PolynomialRing(QQ)
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:

```
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
sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
sage: I = F*[x*y+y*z,x^2+x*y-y*x-y^2]*F
sage: Q = F.quo(I)
sage: is_QuotientRing(Q)
True
sage: is_QuotientRing(F)
False
```

4.2 Quotient Ring Elements

AUTHORS:

• William Stein

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

Bases: sage.structure.element.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
(continues on next page)
```
Quotient of Univariate Polynomial Ring in \( x \) over Integer Ring by the ideal \( \langle x^2 + 3x + 4, x^2 + 1 \rangle \)

\[
\text{sage: } v = S.gens(); v \\
(xbar,)
\]

\[
\text{sage: } \text{loads}(v[0].\text{dumps()}) == v[0] \\
\text{True}
\]

\[
\text{sage: } R.<x,y> = \text{PolynomialRing}(\text{QQ}, 2) \\
\text{sage: } S = R.\text{quo}(x^2 + y^2); S \\
\text{Quotient of Multivariate Polynomial Ring in } x, y \text{ over Rational Field by the ideal } \\
\langle x^2 + y^2 \rangle
\]

\[
\text{sage: } S.\text{gens()} \\
(xbar, ybar)
\]

We name each of the generators.

\[
\text{sage: } S.<a,b> = R.\text{quotient}(x^2 + y^2) \\
\text{sage: } a \\
a \\
\text{sage: } b \\
b \\
\text{sage: } a^2 + b^2 == 0 \\
\text{True} \\
\text{sage: } b.\text{lift()} \\
y \\
\text{sage: } (a^3 + b^2).\text{lift()} \\
-x*y^2 + y^2
\]

**is_unit()**

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

TODO: This is not fully implemented, as illustrated in the example below. So far, self is determined to be unit only if its representation in the cover ring \( R \) is also a unit.

**EXAMPLES:**

\[
\text{sage: } R.<x,y> = \text{QQ}[]; S.<a,b> = R.\text{quo}(1 - x*y); \text{type}(a) \\
\text{<class 'sage.rings.quotient_ring.QuotientRing_generic_with_category.element_} \\
\text{class'>} \\
\text{sage: } a*b \\
1 \\
\text{sage: } a.\text{is_unit()} \\
\text{Traceback (most recent call last):} \\
... \\
\text{NotImplementedError} \\
\text{sage: } S(1).\text{is_unit()} \\
\text{True}
\]

**lc()**

Return the leading coefficient of this quotient ring element.

**EXAMPLES:**

\[
\text{sage: } R.<x,y,z> = \text{PolynomialRing}(\text{GF}(7), 3, \text{order='lex'}) \\
\text{sage: } I = \text{sage.rings.ideal.FieldIdeal}(R)
\]
lift()
If self is an element of $R/I$, then return self as an element of $R$.

EXAMPLES:

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

lm()
Return the leading monomial of this quotient ring element.

EXAMPLES:

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

lt()
Return the leading term of this quotient ring element.

EXAMPLES:

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

monomials()
Return the monomials in self.

OUTPUT:
A list of monomials.

EXAMPLES:

```python
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)
<class 'sage.rings.quotient_ring.QuotientRing_generic_with_category.element_class'>
sage: a.monomials()
[a]
sage: (a+a*b).monomials()
[a*b, a]
```
reduce \( G \)
Reduce this quotient ring element by a set of quotient ring elements \( G \).

INPUT:

- \( G \) - a list of quotient ring elements

EXAMPLES:

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

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.quotient(x^2 + y^2); type(a)
<class 'sage.rings.quotient_ring.QuotientRing_generic_with_category.element_class'>
sage: a.variables()  
(a,)
sage: b.variables()  
(b,)
sage: s = a^2 + b^2 + 1; s
1
sage: s.variables()  
()
sage: (a+b).variables()  
(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'))
Fraction Field of Univariate Polynomial Ring in x over Rational Field
```
```
Dividing elements often implicitly creates elements of the fraction field:

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

The input must be an integral domain:

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

---

**class** `sage.rings.fraction_field.FractionFieldEmbedding`

**Bases:** `sage.structure.coerce_maps.DefaultConvertMap_unique`

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

**EXAMPLES:**

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

is_injective()  

Return whether this map is injective.

**EXAMPLES:**

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

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

is_surjective()  

Return whether this map is surjective.

**EXAMPLES:**

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

section()  

Return a section of this map.

**EXAMPLES:**

```python
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
```

```python
class sage.rings.fraction_field.FractionFieldEmbeddingSection
    Bases: sage.categories.map.Section

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

    EXAMPLES:

```
```
ring_of_integers()
Return the ring of integers in this fraction field.

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

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

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

characteristic()
Return the characteristic of this fraction field.

EXAMPLES:
```
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()
5
```

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

gen(i=0)
Return the i-th generator of self.

EXAMPLES:
```
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:

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:

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

ngens()
This is the same as for the parent object.

EXAMPLES:

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

random_element(*args, **kwds)
Return a random element in this fraction field.

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

EXAMPLES:

sage: F = ZZ['x'].fraction_field()
sage: F.random_element()  # random
(2*x - 8)/(-x^2 + x)
sage: f = F.random_element(degree=5)
sage: f.numerator().degree()
5
sage: f.denominator().degree()
5

ring()
Return the ring that this is the fraction field of.

EXAMPLES:

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

some_elements()
Return some elements in this field.

EXAMPLES:

sage: R.<x> = QQ[]
sage: R.fraction_field().some_elements()
[0, 1, x, 2*x, x/(x^2 + 2*x + 1), 1/x^2, ...
(2*x^2 + 2)/x^3,
(2*x^2 + 2)/(x^2 - 1), 2]

sage.rings.fraction_field.is_FractionField(x)
Test whether or not x inherits from FractionField_generic.

EXAMPLES:

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

5.2 Fraction Field Elements

AUTHORS:

• William Stein (input from David Joyner, David Kohel, and Joe Wetherell)

• Sebastian Pancratz (2010-01-06): Rewrite of addition, multiplication and derivative to use Henrici’s algorithms [Hor1972]
class sage.rings.fraction_field_element.FractionFieldElement
Bases: sage.structure.element.FieldElement

EXAMPLES:

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

denominator()
Return the denominator of self.

EXAMPLES:

```
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:

```
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:

```
sage: R.<t> = QQ[]
sage: (1/t).is_square()  # (continues on next page)
```
True
\begin{verbatim}
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
\end{verbatim}
\begin{verbatim}
sage: (4*(1+t)^4/t^6).is_square(root=True)
(True, (2*t^2 + 4*t + 2)/t^3)
\end{verbatim}
\begin{verbatim}
sage: (2*(1+t)^4/t^6).is_square(root=True)
(False, None)
\end{verbatim}
\begin{verbatim}
sage: R.<x> = QQ[]
sage: a = 2*(x+1)^2 / (2*(x-1)^2); a
(x^2 + 2*x + 1)/(x^2 - 2*x + 1)
sage: a.is_square()
True
sage: (0/x).is_square()
True
\end{verbatim}

is_zero()
Return True if this element is equal to zero.

\textbf{EXAMPLES:}
\begin{verbatim}
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
\end{verbatim}

nth_root \((n)\)
Return a \(n\)-th root of this element.

\textbf{EXAMPLES:}
\begin{verbatim}
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)
Traceback (most recent call last):
  ... ValueError: not a 2nd power
\end{verbatim}

numerator()
Return the numerator of self.
EXAMPLES:

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

**reduce()**
Reduce this fraction.

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

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

EXAMPLES:

```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`

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()
```

(continues on next page)
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:

```
sage: R.<x> = QQ[]
sage: (2 + 2*x) / (4*x) # indirect doctest
(1/2*x + 1/2)/x
```

Compare with:

```
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:**

```
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()
[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 `FractionFieldElement`.

**EXAMPLES:**

```
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 `FractionFieldElement` objects (and subclasses).

**EXAMPLES:**

```
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 FractionFieldElement pickles.

EXAMPLES:

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 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[['u','v']]
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)
O(7^6)
sage: 1/3 + O(7^6)
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 $p$: 
sage: a = O(11^-32); a
O(11^-32)
sage: a.parent()
11-adic Field with capped relative precision 20

There are problems if you add a rational with very negative valuation to an \(O\)-Term:

\[
\text{sage: } 11^{-12} + O(11^{15})
\]
\[
11^{-12} + O(11^8)
\]

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:

\[
\text{sage: } K = \mathbb{Q}_p(11, 30)
\]

\[
\text{sage: } K(11^{-12}) + O(11^{15})
\]
\[
11^{-12} + O(11^{15})
\]
\[
\text{sage: } K(11^{-12}, absprec = 15)
\]
\[
11^{-12} + O(11^{15})
\]
\[
\text{sage: } K(11^{-12}, 15)
\]
\[
11^{-12} + O(11^{15})
\]

We can also work with asymptotic expansions:

\[
\text{sage: } A.<n> = AsymptoticRing(growth_group='\mathbb{Q}^n \ast n^{\mathbb{Q}} \ast \log(n)^\mathbb{Q}', coefficient_ring=\mathbb{Q}); A
\]

Asymptotic Ring \(\mathbb{Q}^n \ast n^{\mathbb{Q}} \ast \log(n)^\mathbb{Q} \ast \text{Signs}^n\) over Rational Field

\[
\text{sage: } O(n)
\]

\[
O(n)
\]

### 6.2 Signed and Unsigned Infinities

The unsigned infinity “ring” is the set of two elements

1. \(\infty\)
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:

```
sage: oo
+Infinity
sage: oo is InfinityRing.0
True
sage: oo is UnsignedInfinityRing.0
False
sage: oo == UnsignedInfinityRing.0
True
```

We do arithmetic:

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

We make 1 / unsigned_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
sage: unsigned_oo/unsigned_oo
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 $\infty$ is predefined as a shorthand for $+\infty$:

```
sage: oo
+Infinity
```

We compare finite and infinite elements:

```
sage: 5 < oo
True
sage: P(-5) < P(5)
True
sage: P(2) < P(3)
False
sage: -oo < oo
True
```

We can do more arithmetic than in the unsigned infinity ring:

```
sage: 2 * oo
+Infinity
sage: -2 * oo
-Infinity
sage: 1 - oo
-Infinity
sage: 1 / oo
0
sage: -1 / oo
0
```

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

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

If we try to subtract infinities or multiply infinity by zero we still get an error:
Signed infinity can also be represented by RR / RDF elements. But unsigned infinity cannot:

```
sage: oo in RR, oo in RDF
(True, True)
sage: unsigned_infinity in RR, unsigned_infinity in RDF
(False, False)
```

class sage.rings.infinity.AnInfinity
Bases: object

```
lcm(x)
   Return the least common multiple of oo and x, which is by definition oo unless x is 0.

   EXAMPLES:
   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
```

class sage.rings.infinity.FiniteNumber(parent, x)
Bases: sage.structure.element.RingElement

Initialize self.

```
sqrt()
   EXAMPLES:
   sage: InfinityRing(7).sqrt()
   A positive finite number
   sage: InfinityRing(0).sqrt()
   Zero
   sage: InfinityRing(-.001).sqrt()
   Traceback (most recent call last):
   ...
   SignError: cannot take square root of a negative number
```

sage.rings.infinity.InfinityRing = The Infinity Ring

6.2. Signed and Unsigned Infinities
class sage.rings.infinity.InfinityRing_class
    Bases: sage.misc.fast_methods.Singleton, sage.rings.ring.Ring

    Initialize self.

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

    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 (parent=The Unsigned Infinity Ring)
    Bases: sage.rings.infinity._uniq, sage.structure.element.RingElement

    Initialize self.

    EXAMPLES:
sage: sage.rings.infinity.LessThanInfinity() is UnsignedInfinityRing(5)
True

class sage.rings.infinity.MinusInfinity
    Bases: sage.rings.infinity._uniq, sage.rings.infinity.AnInfinity, sage.structure.element.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
    Bases: sage.rings.infinity._uniq, sage.rings.infinity.AnInfinity, sage.structure.element.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: exceptions.ArithmeticError

    Sign error exception.

class sage.rings.infinity.UnsignedInfinity
    Bases: sage.rings.infinity._uniq, sage.rings.infinity.AnInfinity, sage.structure.element.InfinityElement

    Initialize self.

sage.rings.infinity.UnsignedInfinityRing = The Unsigned Infinity Ring

class sage.rings.infinity.UnsignedInfinityRing_class
    Bases: sage.misc.fast_methods.Singleton, sage.rings.ring.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

6.2. Signed and Unsigned Infinities
The "generator" of `self` is the infinity object.

**EXAMPLES:**

```python
sage: UnsignedInfinityRing.gen()
Infinity
sage: UnsignedInfinityRing.gen(1)
Traceback (most recent call last):
  ...  
IndexError: UnsignedInfinityRing only has one generator
```

The "generator" of `self` is the infinity object.

**EXAMPLES:**

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

This is the element that represents a finite value.

**EXAMPLES:**

```python
sage: UnsignedInfinityRing.less_than_infinity()
A number less than infinity
sage: UnsignedInfinityRing(5) is UnsignedInfinityRing.less_than_infinity()
True
```

The unsigned infinity ring has one "generator."

**EXAMPLES:**

```python
sage: UnsignedInfinityRing.ngens()
1
sage: len(UnsignedInfinityRing.gens())
1
```

This is a type check for infinity elements.

**EXAMPLES:**

```python
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
```

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:

```python
sage: from sage.rings.infinity import test_comparison
sage: rings = [ZZ, QQ, RR, RealField(200), RDF, RLF, AA, 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 Algebraic Real Field
testing Real Interval Field with 53 bits of precision
```

Comparison with number fields does not work:

```python
sage: K.<sqrt3> = NumberField(x^2-3)
sage: (-oo < 1+sqrt3) and (1+sqrt3 < oo)  # known bug
False
```

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

```python
sage: test_comparison(SR)  # known bug
Traceback (most recent call last):
  ...
AssertionError: testing -1000.0 in Symbolic Ring: id = ...
```

```
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 trac ticket #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 trac ticket #14045 is fixed:

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

EXAMPLES:

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)

6.3 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:
```python
sage: from numpy import isscalar
sage: isscalar(3.141)
True
sage: isscalar(4/17)
True
```
7.1 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:

\begin{verbatim}
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
\end{verbatim}

The method $\texttt{gens()}$ returns the generators of this module:

\begin{verbatim}
sage: A.<x,y,z> = QQ[]
sage: M = A.derivation_module()
sage: M.gens()
(d/dx, d/dy, d/dz)
\end{verbatim}

We can combine them in order to create all derivations:

\begin{verbatim}
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
\end{verbatim}

and now play with them:
Alternatively we can use the method `derivation()` of the ring $A$ to create derivations:

```python
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
```

Sage knows moreover that $M$ is a Lie algebra:

```python
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
```

Computations of Lie brackets are implemented as well:

```python
sage: Dx.bracket(Dy)
0
sage: d.bracket(Dx)
-2*x*d/dz
```

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

```python
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
```

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:

```python
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
```

Now we use this ring homomorphism to define a structure of $A$-algebra on $Q$ and then build the following module of derivations:
Module of derivations from Multivariate Polynomial Ring in \(x, y, z\) over Rational Field to Rational Field

\[
\text{sage: } M = A.derivation_module(ev)
\]
\[
\text{sage: } M.gens()
\]
\[(d/dx, d/dy, d/dz)\]

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

\[
\text{sage: } D_x = M.gen(0)
\]
\[
\text{sage: } D_y = M.gen(1)
\]
\[
\text{sage: } D_z = M.gen(2)
\]
\[
\text{sage: } f = x^2 + y^2 + z^2
\]
\[
\text{sage: } D_x(f) \quad \# = 2*x \text{ evaluated at } (0,1,2)
\]
0
\[
\text{sage: } D_y(f) \quad \# = 2*y \text{ evaluated at } (0,1,2)
\]
2
\[
\text{sage: } D_z(f) \quad \# = 2*z \text{ evaluated at } (0,1,2)
\]
4

Twisted derivations are handled similarly:

\[
\text{sage: } \theta = B.hom([B(y), B(z), B(x)])
\]
\[
\text{sage: } \theta
\]
Ring endomorphism of Fraction Field of Multivariate Polynomial Ring in \(x, y, z\) over Rational Field
\[
\text{Defn: } x |--> y
\]
\[
y |--> z
\]
\[
z |--> x
\]
\[
\text{sage: } M = B.derivation_module(twist=\theta)
\]
\[
\text{sage: } M
\]
Module of twisted derivations over Fraction Field of Multivariate Polynomial Ring in \(x, y, z\) over Rational Field (twisting morphism: \(x \mapsto y, y \mapsto z, z \mapsto x\))

Over a field, one proves that every \(\theta\)-derivation is a multiple of \(\theta - id\), so that:

\[
\text{sage: } d = M.gen(); d
\]
\[
[x \mapsto y, y \mapsto z, z \mapsto x] - id
\]

and then:

\[
\text{sage: } d(x)
\]
\[-x + y\]
\[
\text{sage: } d(y)
\]
\[-y + z\]
\[
\text{sage: } d(z)
\]
\[x - z\]
\[
\text{sage: } d(x + y + z)
\]
0

AUTHOR:

- Xavier Caruso (2018-09)

**class** **sage.rings.derivation.RingDerivation**

Bases: **sage.structure.element.ModuleElement**

An abstract class for twisted and untwisted derivations over commutative rings.
codomain()  
Return the codomain of this derivation.

EXAMPLES:

```python
sage: R.<x> = QQ[]
sage: f = R.derivation(); f
d/dx
sage: f.codomain()
Univariate Polynomial Ring in x over Rational Field
sage: f.codomain() is R
True
```

```python
sage: S.<y> = R[]
sage: M = R.derivation_module(S)
sage: M.random_element().codomain()
Univariate Polynomial Ring in y over Univariate Polynomial Ring in x over Rational Field
sage: M.random_element().codomain() is S
True
```

domain()  
Return the domain of this derivation.

EXAMPLES:

```python
sage: R.<x,y> = ZZ[]
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
Bases: sage.modules.module.Module, sage.structure.unique_representation.UniqueRepresentation

A class for modules of derivations over a commutative ring.

basis()  
Return a basis of this module of derivations.

EXAMPLES:

```python
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module()
sage: M.basis()
Family (d/dx, d/dy)
```

codomain()  
Return the codomain of the derivations in this module.

EXAMPLES:

```python
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module(); M
Module of derivations over Multivariate Polynomial Ring in x, y over Integer Ring
```
**defining_morphism()**

Return the morphism defining the structure of algebra of the codomain over the domain.

**EXAMPLES:**

```sage
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
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module(); M
Module of derivations over Multivariate Polynomial Ring in x, y over Integer Ring
sage: M.domain()
Multivariate Polynomial Ring in x, y over Integer Ring
```

**dual_basis()**

Return the dual basis of the canonical basis of this module of derivations (which is that returned by the method `basis()`).

**Note:** The dual basis of \((d_1, \ldots, d_n)\) is a family \((x_1, \ldots, x_n)\) of elements in the domain such that \(d_i(x_i) = 1\) and \(d_i(x_j) = 0\) if \(i \neq j\).

**EXAMPLES:**

```sage
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module()
sage: M.basis()
Family (d/dx, d/dy)
sage: M.dual_basis()
Family (x, y)
```
gen \( (n=0) \)
Return the \( n \)-th generator of this module of derivations.

**INPUT:**

- \( n \) – an integer (default: 0)

**EXAMPLES:**

```python
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module(); M
```
```
Module of derivations over Multivariate Polynomial Ring in x, y over Integer
```
```
˓→Ring
```
```
sage: M.gen()
d/dx
```
```
sage: M.gen(1)
d/dy
```
```
gens ()
Return the generators of this module of derivations.

**EXAMPLES:**

```python
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module(); M
```
```
Module of derivations over Multivariate Polynomial Ring in x, y over Integer
```
```
˓→Ring
```
```
sage: M.gens()
```
```
(d/dx, d/dy)
```
```
We check that, for a nontrivial twist over a field, the module of twisted derivation is a vector space of
dimension 1 generated by \( \text{twist} - \text{id} \):

```python
sage: K = R.fraction_field()
sage: theta = K.hom([K(y),K(x)])
sage: M = K.derivation_module(twist=theta); M
```
```
Module of twisted derivations over Fraction Field of Multivariate Polynomial
```
```
Ring in x, y over Integer Ring (twisting morphism: x |--> y, y |--> x)
```
```
sage: M.gens()
```
```
([x |--> y, y |--> x] - id,) 
```
```
gens ()
Return the number of generators of this module of derivations.

**EXAMPLES:**

```python
sage: R.<x,y> = ZZ[]
sage: M = R.derivation_module(); M
```
```
Module of derivations over Multivariate Polynomial Ring in x, y over Integer
```
```
˓→Ring
```
```
sage: M.ngens()
```
```
2
```
```
Indeed, generators are:

```python
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 \( \text{twist} - \text{id} \):
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.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:
class sage.rings.derivation.RingDerivationWithTwist_generic (parent, scalar=0)
Bases: sage.rings.derivation.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.

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

EXAMPLES:

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:

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

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

### class `sage.rings.derivation.RingDerivationWithoutTwist`

An abstract class for untwisted derivations.

#### 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> = QQ[]
sage: M = R.derivation_module()
sage: M.basis()
Family (d/dx, d/dy)
sage: R.derivation(x).list()
[1, 0]
sage: R.derivation(y).list()
[0, 1]
sage: f = x*R.derivation(x) + y*R.derivation(y); f
x*d/dx + y*d/dy
sage: f.list()
[x, y]
```

#### monomial_coefficients()  

Return dictionary of nonzero coordinates (on the canonical basis) of this derivation.

### 7.1. Derivations

105
More precisely, this returns a dictionary whose keys are indices of basis elements and whose values are the corresponding coefficients.

**EXAMPLES:**

```python
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:**

```python
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*:

```python
defx = Dx.postcompose(ev)
dy = Dy.postcompose(ev)
def = x^2 + y^2
def(dx(f)
0
def(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*:

```python
defx.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:

```python
defx.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: R.<x,y> = GF(5)[]
sage: Dx = R.derivation(x)
sage: Dx.pth_power()
0
sage: (x*Dx).pth_power()
x^d/dx
sage: (x^6*Dx).pth_power()
x^26*d/dx
sage: Dy = R.derivation(y)
sage: (x*Dx + y*Dy).pth_power()
x^d/dx + y*d/dy
```
An error is raised if the domain has characteristic zero:

```python
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:

```python
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`

```
Bases: sage.rings.derivation.RingDerivationWithoutTwist_wrapper

This class handles derivations over fraction fields.
```

class `sage.rings.derivation.RingDerivationWithoutTwist_function`

```
Bases: sage.rings.derivation.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)[]
sage: M = R.derivation_module()
sage: M.basis()
Family (d/dx, d/dy)
sage: R.derivation(x).list()
[1, 0]
sage: R.derivation(y).list()
```

(continues on next page)
class sage.rings.derivation.RingDerivationWithoutTwist_quotient (parent, arg=None)

This class handles derivations over quotient rings.

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

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:

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

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 \( \mathbb{Z}, \mathbb{Q} \), a finite field, etc.)

is_zero()

Return True if this derivation vanishes.

EXAMPLES:

list()

Return the list of coefficient of this derivation on the canonical basis.

7.1. Derivations 109
EXAMPLES:

```python
sage: M = QQ.derivation_module()
sage: M().list()
[]
```
CHAPTER
EIGHT

INDICES AND TABLES

• Index
• Module Index
• Search Page
r
sage.rings.big_oh, 85
sage.rings.derivation, 97
sage.rings.fraction_field, 73
sage.rings.fraction_field_element, 78
sage.rings.homset, 54
sage.rings.ideal, 25
sage.rings.ideal_monoid, 39
sage.rings.infinity, 86
sage.rings.morphism, 43
sage.rings.noncommutative_ideals, 39
sage.rings.numbers_abc, 94
sage.rings.quotient_ring, 57
sage.rings.quotient_ring_element, 68
sage.rings.ring, 1
INDEX

A
absolute_norm() (sage.rings.ideal.Ideal_generic method), 28
Algebra (class in sage.rings.ring), 2
algebraic_closure() (sage.rings.ring.Field method), 9
ambient() (sage.rings.quotient_ring.QuotientRing_nc method), 61
AnInfinity (class in sage.rings.infinity), 89
apply_morphism() (sage.rings.ideal.Ideal_generic method), 28
associated_primes() (sage.rings.ideal.Ideal_generic method), 28

B
base_extend() (sage.rings.ring.Ring method), 14
base_ring() (sage.rings.fraction_field.FractionField_generic method), 76
base_ring() (sage.rings.ideal.Ideal_generic method), 29
basis() (sage.rings.derivation.RingDerivationModule method), 100

class

category() (sage.rings.ideal.Ideal_generic method), 29
category() (sage.rings.ring.Ring method), 14
characteristic() (sage.rings.fraction_field.FractionField_generic method), 76
characteristic() (sage.rings.quotient_ring.QuotientRing_nc method), 61
characteristic() (sage.rings.ring.Algebra method), 2
class_group() (sage.rings.ring.PrincipalIdealDomain method), 12
class_number() (sage.rings.fraction_field.FractionField_1poly_field method), 75
codomain() (sage.rings.derivation.RingDerivation method), 99
codomain() (sage.rings.derivation.RingDerivationModule method), 100
CommutativeAlgebra (class in sage.rings.ring), 2
CommutativeRing (class in sage.rings.ring), 3
construction() (sage.rings.fraction_field.FractionField_generic method), 76
construction() (sage.rings.quotient_ring.QuotientRing_nc method), 62
content() (sage.rings.ring.PrincipalIdealDomain method), 12
cover() (sage.rings.quotient_ring.QuotientRing_nc method), 62
cover_ring() (sage.rings.quotient_ring.QuotientRing_nc method), 63
Cyclic() (in module sage.rings.ideal), 25

D
DedekindDomain (class in sage.rings.ring), 7
defining_ideal() (sage.rings.quotient_ring.QuotientRing_nc method), 63
defining_morphism() (sage.rings.derivation.RingDerivationModule method), 101
denominator() (sage.rings.fraction_field_element.FractionFieldElement method), 79
derivation() (sage.rings.derivation.RingDerivation method), 3
derivation_module() (sage.rings.ring.CommutativeRing method), 4
divides() (sage.rings.ideal.Ideal_principal method), 37
divides() (sage.rings.ring.Field method), 9
domain() (sage.rings.derivation.RingDerivation method), 100
domain() (sage.rings.derivation.RingDerivationModule method), 101
dual_basis() (sage.rings.derivation.RingDerivationModule method), 101

element (sage.rings.ideal.Monoid_IdealMonoid_c attribute), 39
Element (sage.rings.quotient_ring.QuotientRing nc attribute), 61
embedded_primes() (sage.rings.ideal.Ideal_generic method), 29
epsilon() (sage.rings.ring.Ring method), 15
EuclideanDomain (class in sage.rings.ring), 8
extension() (sage.rings.ring.CommutativeRing method), 5

f
Field (class in sage.rings.ring), 9
FieldIdeal() (in module sage.rings.ideal), 25
FiniteNumber (class in sage.rings.infinity), 89
fraction_field() (sage.rings.infinity.InfinityRing_class method), 90
fraction_field() (sage.rings.infinity.UnsignedInfinityRing_class method), 91
fraction_field() (sage.rings.ring.CommutativeRing method), 6
fraction_field() (sage.rings.ring.Field method), 9
FractionField() (in module sage.rings.fraction_field), 73
FractionField_Lpoly_field (class in sage.rings.fraction_field), 75
FractionField_generic (class in sage.rings.fraction_field), 76
FractionFieldElement (class in sage.rings.fraction_field), 78
FractionFieldLpoly_field (class in sage.rings.fraction_field_element), 81
FractionFieldEmbedding (class in sage.rings.fraction_field), 74
FractionFieldEmbeddingSection (class in sage.rings.fraction_field), 75
frobenius_endomorphism() (sage.rings.ring.CommutativeRing method), 6
FrobeniusEndomorphism_generic (class in sage.rings.morphism), 48
function_field() (sage.rings.fraction_field.FractionField_Lpoly_field method), 75

G
gcd() (sage.rings.ideal.Ideal_pid method), 35
gcd() (sage.rings.ring.PrincipalIdealDomain method), 13
gen() (sage.rings.derivation.RingDerivationModule method), 101
gen() (sage.rings.fraction_field.FractionField_generic method), 76
gen() (sage.rings.ideal.Ideal_generic method), 30
gen() (sage.rings.ideal.Ideal_principal method), 37
gen() (sage.rings.infinity.InfinityRing_class method), 90
gen() (sage.rings.infinity.UnsignedInfinityRing_class method), 91
gen() (sage.rings.quotient_ring.QuotientRing_nc method), 63
gens() (sage.rings.derivation.RingDerivationModule method), 102
gens() (sage.rings.ideal.Ideal_generic method), 30
gens() (sage.rings.infinity.InfinityRing_class method), 90

116
Index 117

gens() (sage.rings.infinity.UnsignedInfinityRing_class method), 92

gens_reduced() (sage.rings.ideal.Ideal_generic method), 30

H

has_coerce_map_from() (sage.rings.homset.RingHomset_generic method), 55

has_standard_involution() (sage.rings.ring.Algebra method), 2

I

Ideal() (in module sage.rings.ideal), 26

ideal() (sage.rings.quotient_ring.QuotientRing nc method), 64

ideal() (sage.rings.ring.Field method), 10

ideal() (sage.rings.ring.Ring method), 15

Ideal_fractional (class in sage.rings.ideal), 27

Ideal_generic (class in sage.rings.ideal), 28

ideal_monoid() (sage.rings.ring.CommutativeRing method), 6

ideal_monoid() (sage.rings.ring.Ring method), 16

Ideal_nc (class in sage.rings.noncommutative_ideals), 40

Ideal_pid (class in sage.rings.ideal), 34

Ideal_principal (class in sage.rings.ideal), 37

IdealMonoid() (in module sage.rings.ideal_monoid), 39

IdealMonoid_c (class in sage.rings.ideal_monoid), 39

IdealMonoid_nc (class in sage.rings.noncommutative_ideals), 40

im_gens() (sage.rings.morphism.RingHomomorphism_im_gens method), 53

InfinityRing (in module sage.rings.infinity), 89

InfinityRing_class (class in sage.rings.infinity), 89

integral_closure() (sage.rings.ring.DedekindDomain method), 7

integral_closure() (sage.rings.ring.Field method), 10

IntegralDomain (class in sage.rings.ring), 11

inverse_image() (sage.rings.morphism.RingHomomorphism method), 49

is_commutative() (sage.rings.infinity.UnsignedInfinityRing_class method), 90

is_commutative() (sage.rings.quotient_ring.QuotientRing nc method), 64

is_commutative() (sage.rings.ring.CommutativeAlgebra method), 2

is_commutative() (sage.rings.ring.CommutativeRing method), 6

is_commutative() (sage.rings.ring.Ring method), 16

is_exact() (sage.rings.fraction_field.FractionField_generic method), 77

is_exact() (sage.rings.ring.Ring method), 16

is_field() (sage.rings.fraction_field.FractionField_generic method), 77

is_field() (sage.rings.ring.Ring method), 16

is_field() (sage.rings.quotient_ring.QuotientRing nc method), 64

is_field() (sage.rings.ring.Field method), 10

is_field() (sage.rings.ring.IntegralDomain method), 11

is_field() (sage.rings.ring.Ring method), 17

is_finite() (sage.rings.fraction_field.FractionField_generic method), 77

is_FractionField() (in module sage.rings.fraction_field), 78

is_FractionFieldElement() (in module sage.rings.fraction_field_element), 82

is_Ideal() (in module sage.rings.ideal), 38

is_Infinite() (in module sage.rings.infinity), 92

is_injective() (sage.rings.fraction_field.FractionFieldEmbedding method), 74

is_integral() (sage.rings.fraction_field_element.FractionFieldElement_1poly_field method), 81

is_integral_domain() (sage.rings.quotient_ring.QuotientRing nc method), 65

is_integral_domain() (sage.rings.ring.IntegralDomain method), 11
is_integral_domain() (sage.rings.ring.Ring method), 17
is_integrally_closed() (sage.rings.ring.DedekindDomain method), 7
is_integrally_closed() (sage.rings.ring.Field method), 10
is_integrally_closed() (sage.rings.ring.IntegralDomain method), 11
is_maximal() (sage.rings.ideal.Ideal_generic method), 30
is_maximal() (sage.rings.ideal.Ideal_pid method), 35
is_noetherian() (sage.rings.quotient_ring.QuotientRing_nc method), 65
is_noetherian() (sage.rings.ring.DedekindDomain method), 8
is_noetherian() (sage.rings.ring.Field method), 10
is_noetherian() (sage.rings.ring.NoetherianRing method), 12
is_noetherian() (sage.rings.ring.PrincipalIdealDomain method), 14
is_noetherian() (sage.rings.ring.Ring method), 18
is_one() (sage.rings.fraction_field_element.FractionFieldElement method), 79
is_prime() (sage.rings.ideal.Ideal_generic method), 31
is_prime() (sage.rings.ideal.Ideal_pid method), 35
is_principal_field() (in module sage.rings.quotient_ring), 68
is_Ring() (in module sage.rings.ring), 23
is_Ring() (sage.rings.ring.Ring method), 18
is_RingHomomorphism() (in module sage.rings.morphism), 54
is_RingHomset() (in module sage.rings.homset), 56
is_square() (sage.rings.fraction_field_element.FractionFieldElement method), 79
is_subring() (sage.rings.ring.Ring method), 19
is_surjective() (sage.rings.fraction_field.FractionFieldEmbedding method), 74
is_trivial() (sage.rings.ideal.Ideal_generic method), 32
is_unit() (sage.rings.quotient_ring_element.QuotientRingElement method), 69
is_zero() (sage.rings.derivation.RingDerivationWithoutTwist method), 105
is_zero() (sage.rings.derivation.RingDerivationWithoutTwist_function method), 108
is_zero() (sage.rings.derivation.RingDerivationWithoutTwist_zero method), 109
is_zero() (sage.rings.fraction_field_element.FractionFieldElement method), 80
is_zero() (sage.rings.infinity.InfinityRing_class method), 90

K
Katsura() (in module sage.rings.ideal), 38
kernel() (sage.rings.morphism.RingHomomorphism_cover method), 50
krull_dimension() (sage.rings.ring.CommutativeRing method), 7
krull_dimension() (sage.rings.ring.DedekindDomain method), 8
krull_dimension() (sage.rings.ring.Field method), 10

L
lc() (sage.rings.quotient_ring_element.QuotientRingElement method), 69
lcm() (sage.rings.infinity.AnInfinity method), 89
less_than_infinity() (sage.rings.infinity.UnsignedInfinityRing_class method), 92
LessThanInfinity (class in sage.rings.infinity), 90
lift() (sage.rings.morphism.RingHomomorphism method), 49
lift() (sage.rings.quotient_ring.QuotientRing_nc method), 65
lift() (sage.rings.quotient_ring_element.QuotientRingElement method), 70
lifting_map() (sage.rings.quotient_ring.QuotientRing_nc method), 66
list() (sage.rings.derivation.RingDerivationWithoutTwist method), 105
list() (sage.rings.derivation.RingDerivationWithoutTwist_wrapping function method), 108
list() (sage.rings.derivation.RingDerivationWithoutTwist_zero method), 109
list() (sage.rings.derivation.RingDerivationWithTwist_generic method), 104
lm() (sage.rings.quotient_ring_element.QuotientRingElement method), 70
lt() (sage.rings.quotient_ring_element.QuotientRingElement method), 70

M
make_element() (in module sage.rings.fraction_field_element), 82
make_element_old() (in module sage.rings.fraction_field_element), 83
maximal_order() (sage.rings.fraction_field.FractionField_Frac_field method), 75
minimal_associated_primes() (sage.rings.ideal.Ideal_generic method), 32
MinusInfinity (class in sage.rings.infinity), 91
monomial_coefficients() (sage.rings.derivation.RingDerivationWithoutTwist method), 105
monomials() (sage.rings.quotient_ring_element.QuotientRingElement method), 70
morphism_from_cover() (sage.rings.morphism.RingHomomorphism_from_quotient method), 53

N
natural_map() (sage.rings.homset.RingHomset_generic method), 55
ngens() (sage.rings.derivation.RingDerivationModule method), 102
ngens() (sage.rings.fraction_field.FractionField_generic method), 77
ngens() (sage.rings.ideal.Ideal_generic method), 33
ngens() (sage.rings.infinity.InfinityRing_class method), 90
ngens() (sage.rings.infinity.UnsignedInfinityRing_class method), 92
ngens() (sage.rings.quotient_ring_element.QuotientRingElement method), 67
NoetherianRing (class in sage.rings.ring), 12
nth_root() (sage.rings.ideal.Ideal_generic method), 33
numerator() (sage.rings.fraction_field_element.FractionFieldElement method), 80

O
O() (in module sage.rings.big_oh), 85
one() (sage.rings.ring.Ring method), 19
order() (sage.rings.ring.Ring method), 19

P
parameter() (sage.rings.ring.EuclideanDomain method), 9
PlusInfinity (class in sage.rings.infinity), 91
postcompose() (sage.rings.derivation.RingDerivationWithoutTwist method), 106
postcompose() (sage.rings.derivation.RingDerivationWithTwist_generic method), 104
power() (sage.rings.morphism.FrobeniusEndomorphism_generic method), 49
precompose() (sage.rings.derivation.RingDerivationWithoutTwist method), 106
precompose() (sage.rings.derivation.RingDerivationWithTwist_generic method), 104
primary_decomposition() (sage.rings.ideal.Ideal_generic method), 33
principal_subfield() (sage.rings.ring.Field method), 11
principal_ideal() (sage.rings.ring.Ring method), 19
PrincipalIdealDomain (class in sage.rings.ring), 12
pth_power() (sage.rings.derivation.RingDerivationWithoutTwist method), 107
pushforward() (`sage.rings.morphism.RingHomomorphism method`), 49
Python Enhancement Proposals

PEP 3141, 94

Q

quo() (`sage.rings.ring.Ring method`), 19
quotient() (`sage.rings.ring.Ring method`), 20
quotient_ring() (`sage.rings.ring.Ring method`), 20
QuotientRing() (in module `sage.rings.quotient_ring`), 58
QuotientRing_generic (class in `sage.rings.quotient_ring`), 60
QuotientRing_nc (class in `sage.rings.quotient_ring`), 60
QuotientRingElement (class in `sage.rings.quotient_ring_element`), 68

R

random_element() (`sage.rings.derivation.RingDerivationModule method`), 103
random_element() (`sage.rings.fraction_field.FractionField_generic method`), 77
random_element() (`sage.rings.ideal.Ideal_generic method`), 33
random_element() (`sage.rings.ring.Ring method`), 21
reduce() (`sage.rings.fraction_field_element.FractionFieldElement method`), 81
reduce() (`sage.rings.fraction_field_element.FractionFieldElement_Ipoly_field method`), 82
reduce() (`sage.rings.ideal.Ideal_generic method`), 34
reduce() (`sage.rings.ideal.Ideal_pid method`), 36
reduce() (`sage.rings.quotient_ring_element.QuotientRingElement method`), 71
register_sage_classes() (in module `sage.rings.numbers_abc`), 94
residue_field() (in module `sage.rings.ideal`), 36
retract() (`sage.rings.quotient_ring.QuotientRing_nc method`), 67
Ring (class in `sage.rings.ring`), 14
ring() (`sage.rings.fraction_field.FractionField_generic method`), 78
ring() (`sage.rings.ideal.Ideal_generic method`), 34
ring() (`sage.rings.ideal_monoid.IdealMonoid_c method`), 39
ring_of_constants() (`sage.rings.derivation.RingDerivationModule method`), 103
ring_of_integers() (`sage.rings.fraction_field.FractionField_Ipoly_field method`), 75
RingDerivation (class in `sage.rings.derivation`), 99
RingDerivationModule (class in `sage.rings.derivation`), 100
RingDerivationWithoutTwist (class in `sage.rings.derivation`), 105
RingDerivationWithoutTwist_fraction_field (class in `sage.rings.derivation`), 108
RingDerivationWithoutTwist_function (class in `sage.rings.derivation`), 108
RingDerivationWithoutTwist_quotient (class in `sage.rings.derivation`), 109
RingDerivationWithoutTwist_wrapper (class in `sage.rings.derivation`), 109
RingDerivationWithTwist (class in `sage.rings.derivation`), 104
RingHomomorphism (class in `sage.rings.morphism`), 49
RingHomomorphism_coercion (class in `sage.rings.morphism`), 50
RingHomomorphism_cover (class in `sage.rings.morphism`), 50
RingHomomorphism_from_base (class in `sage.rings.morphism`), 50
RingHomomorphism_from_quotient (class in `sage.rings.morphism`), 52
RingHomomorphism_im_gens (class in `sage.rings.morphism`), 53
RingHomset() (in module `sage.rings.homset`), 54
RingHomset_generic (class in `sage.rings.homset`), 54
RingHomset_quo_ring (class in `sage.rings.homset`), 55
RingMap (class in sage.rings.morphism), 53
RingMap_lift (class in sage.rings.morphism), 53

S
sage.rings.big_oh (module), 85
sage.rings.derivation (module), 97
sage.rings.fraction_field (module), 73
sage.rings.fraction_field_element (module), 78
sage.rings.homset (module), 54
sage.rings.ideal (module), 25
sage.rings.ideal_monoid (module), 39
sage.rings.infinity (module), 86
sage.rings.morphism (module), 43
sage.rings.noncommutative_ideals (module), 39
sage.rings.numbers_abc (module), 94
sage.rings.quotient_ring (module), 57
sage.rings.quotient_ring_element (module), 68
sage.rings.ring (module), 1
section() (sage.rings.fraction_field.FractionFieldEmbedding method), 74
side() (sage.rings.noncommutative_ideals.Ideal_nc method), 41
SignError, 91
some_elements() (sage.rings.derivation.RingDerivationModule method), 103
some_elements() (sage.rings.fraction_field.FractionField_generic method), 78
specialization() (sage.rings.fraction_field_element.FractionFieldElement method), 81
sqrt() (sage.rings.infinity.FiniteNumber method), 89
sqrt() (sage.rings.infinity.MinusInfinity method), 91
sqrt() (sage.rings.infinity.PlusInfinity method), 91
support() (sage.rings.fraction_field_element.FractionFieldElement_1poly_field method), 82

T
term_order() (sage.rings.quotient_ring.QuotientRing_nc method), 67
test_comparison () (in module sage.rings.infinity), 92
test_signed_infinity () (in module sage.rings.infinity), 93
twisting_morphism() (sage.rings.derivation.RingDerivationModule method), 103

U
underlying_map() (sage.rings.morphism.RingHomomorphism_from_base method), 52
unit_ideal() (sage.rings.ring.Ring method), 21
UnsignedInfinity (class in sage.rings.infinity), 91
UnsignedInfinityRing (in module sage.rings.infinity), 91
UnsignedInfinityRing_class (class in sage.rings.infinity), 91

V
valuation() (sage.rings.fraction_field_element.FractionFieldElement method), 81
variables() (sage.rings.quotient_ring_element.QuotientRingElement method), 71

Z
zero() (sage.rings.homset.RingHomset_generic method), 55
zero() (sage.rings.ring.Ring method), 21
zero_ideal() (sage.rings.ring.Ring method), 21
zeta() (sage.rings.ring.Ring method), 22
zeta_order() (sage.rings.ring.Ring method), 22