Sage Reference Manual: General Rings, Ideals, and Morphisms

Release 6.7

The Sage Development Team

June 24, 2015

CHAPTER

ONE

RINGS

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

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

The class inheritance hierarchy is:

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

Subclasses of PrincipalIdealDomain are

- EuclideanDomain
- Field
  - FiniteField

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

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

AUTHORS:

- David Harvey (2006-10-16): changed CommutativeAlgebra to derive from CommutativeRing instead of from Algebra.
- David Loeffler (2009-07-09): documentation fixes, added to reference manual.
- Simon King (2011-03-29): Proper use of the category framework for rings.

- Simon King (2011-05-20): Modify multiplication and _ideal_class_ to support ideals of non-commutative rings.

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
<type 'sage.rings.ring.Algebra'>
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 $IdentifyingtheMatrixRing$. 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.

`extension`(*poly*, *name=None*, *names=None*, *embedding=None*)

Algebraically extends self by taking the quotient `self[x] / (f(x))`.

INPUT:

- `poly` – A polynomial whose coefficients are coercible into `self`
- `name` – (optional) name for the root of f

Note: Using this method on an algebraically complete field does *not* return this field; the construction `self[x] / (f(x))` is done anyway.

EXAMPLES:

```
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

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
```

`fraction_field`()

Return the fraction field of `self`.

EXAMPLES:

```
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:

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

We can specify a power:

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

ideal_monoid()

Return the monoid of ideals of this ring.

EXAMPLES:

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

is_commutative()

Return `True`, since this ring is commutative.

EXAMPLES:

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

TESTS:

`krull_dimension` is not implemented for generic commutative rings. Fields and PIDs, with Krull dimension equal to 0 and 1, respectively, have naive implementations of `krull_dimension`. Orders in number fields also have Krull dimension 1:

```
sage: R = CommutativeRing(ZZ)
sage: R.krull_dimension()
Traceback (most recent call last):
...
NotImplementedError
sage: QQ.krull_dimension()
0
sage: ZZ.krull_dimension()
1
sage: type(R); type(QQ); type(ZZ)
<type 'sage.rings.ring.CommutativeRing'>
<class 'sage.rings.rational_field.RationalField_with_category'>
<type 'sage.rings.integer_ring.IntegerRing_class'>
```

All orders in number fields have Krull dimension 1, including non-maximal orders:

```
sage: K.<i> = QuadraticField(-1)
sage: R = K.maximal_order(); R
Maximal Order in Number Field in i with defining polynomial x^2 + 1
sage: R.krull_dimension()
1
```

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

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

is_noetherian()

Return `True` since Dedekind domains are Noetherian.

EXAMPLES:

The integers, **Z**, and rings of integers of number fields are Dedekind domains:

```
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):

```
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:

```
sage: QQ.krull_dimension()
0
sage: T.<x,y> = PolynomialRing(QQ,2); T
Multivariate Polynomial Ring in x, y over Rational Field
sage: T.krull_dimension()
2
sage: U.<x,y,z> = PolynomialRing(ZZ,3); U
Multivariate Polynomial Ring in x, y, z over Integer Ring
sage: U.krull_dimension()
4

sage: K.<i> = QuadraticField(-1)
sage: R = K.order(2*i); R
Order in Number Field in i with defining polynomial x^2 + 1
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):
...
NotImplementedError: Algebraic closures of general fields not implemented.
```

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

Return True if x divides y in this field (usually True in a field!). If coerce is True (the default), first coerce x and y into self.

EXAMPLES:

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

fraction_field()

Return the fraction field of self.

EXAMPLES:

Since fields are their own field of fractions, we simply get the original field in return:

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

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
```

An example where we raise a `NotImplementedError`:

```
sage: R = IntegralDomain(ZZ)
sage: R.is_field()
Traceback (most recent call last):
...
NotImplementedError
```

is_integral_domain(*proof=True*)

Return `True`, since this ring is an integral domain.

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

EXAMPLES:

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

is_integrally_closed()

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

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

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

EXAMPLES:

```
sage: ZZ.is_integrally_closed()
True
sage: QQ.is_integrally_closed()
True
sage: QQbar.is_integrally_closed()
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_integrall
```

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.

is_noetherian()

Return `True` since this ring is Noetherian.

EXAMPLES:

```
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.

class_group()

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

EXAMPLES:

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

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

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

EXAMPLES:

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

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

In a field, any nonzero element is a GCD of any nonempty set of nonzero elements. In previous versions, Sage used to return 1 in the case of the rational field. However, since 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 `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.

TESTS:

This is to test against the bug fixed in trac ticket #9138:

```
sage: R.<x> = QQ[]
sage: R.sum([x,x])
2*x
sage: R.<x,y> = ZZ[]
sage: R.sum([x,y])
x + y
sage: TestSuite(QQ['x']).run(verbose=True)
running ._test_additive_associativity() . . . pass
running ._test_an_element() . . . pass
running ._test_associativity() . . . pass
running ._test_category() . . . pass
running ._test_characteristic() . . . pass
running ._test_distributivity() . . . pass
running ._test_elements() . . .
  Running the test suite of self.an_element()
  running ._test_category() . . . pass
  running ._test_eq() . . . pass
  running ._test_nonzero_equal() . . . pass
  running ._test_not_implemented_methods() . . . pass
  running ._test_pickling() . . . pass
  pass
running ._test_elements_eq_reflexive() . . . pass
running ._test_elements_eq_symmetric() . . . pass
running ._test_elements_eq_transitive() . . . pass
running ._test_elements_neq() . . . pass
running ._test_eq() . . . pass
running ._test_euclidean_degree() . . . pass
running ._test_gcd_vs_xgcd() . . . pass
running ._test_not_implemented_methods() . . . pass
running ._test_one() . . . pass
running ._test_pickling() . . . pass
running ._test_prod() . . . pass
running ._test_quo_rem() . . . pass
running ._test_some_elements() . . . pass
running ._test_zero() . . . pass
running ._test_zero_divisors() . . . pass
sage: TestSuite(QQ['x','y']).run()
sage: TestSuite(ZZ['x','y']).run()
sage: TestSuite(ZZ['x','y']['t']).run()
```

Test agaings another bug fixed in trac ticket #9944:

```
sage: QQ['x'].category()
Join of Category of euclidean domains and Category of commutative algebras over quotient fiel
sage: QQ['x','y'].category()
Join of Category of unique factorization domains and Category of commutative algebras over qu
sage: PolynomialRing(MatrixSpace(QQ,2),'x').category()
Category of algebras over algebras over quotient fields
sage: PolynomialRing(SteenrodAlgebra(2),'x').category()
Category of algebras over graded hopf algebras with basis over Finite Field of size 2
```

```
TESTS::

    sage: Zp(7)._repr_option('element_is_atomic')
    False
    sage: QQ._repr_option('element_is_atomic')
    True
    sage: CDF._repr_option('element_is_atomic')
    False
```

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
```

cardinality()

Return the cardinality of the underlying set.

OUTPUT:

Either an integer or `+Infinity`.

EXAMPLES:

```
sage: Integers(7).cardinality()
7
sage: QQ.cardinality()
+Infinity
```

category()

Return the category to which this ring belongs.

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

EXAMPLES:

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

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

```
sage: I = Integers(15)
sage: I.base_ring() is I
True
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:

```
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:

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

This also works over derived rings:

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

For the symbolic ring, there is no reasonable answer:

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

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

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

INPUT:

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

EXAMPLES:

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

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

Here is an example over a non-commutative ring:

```
sage: A = SteenrodAlgebra(2)
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
```

TESTS:

Make sure that trac ticket #11139 is fixed:

```
sage: R.<x> = QQ[]
sage: R.ideal([])
Principal ideal (0) of Univariate Polynomial Ring in x over Rational Field
sage: R.ideal(())
Principal ideal (0) of Univariate Polynomial Ring in x over Rational Field
sage: R.ideal()
Principal ideal (0) of Univariate Polynomial Ring in x over Rational Field
```

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
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.quo(I)
sage: Q.ideal_monoid()
Monoid of ideals of Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) ov
```

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:

```
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
```

is_field(*proof=True*)

Return `True` if this ring is a field.

INPUT:

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

ALGORITHM:

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

EXAMPLES:

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

This illustrates the use of the `proof` parameter:

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

is_finite()

Return `True` if this ring is finite.

EXAMPLES:

```
sage: QQ.is_finite()
False
sage: GF(2^10,'a').is_finite()
True
sage: R.<x> = GF(7)[]
sage: R.is_finite()
False
sage: S.<y> = R.quo(x^2+1)
```

```
sage: S.is_finite()
True
```

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> = QQ[]
sage: S.<x,y> = R.quo((b^3))
sage: S.is_integral_domain()
False
```

This illustrates the use of the proof parameter:

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

TESTS:

Make sure trac ticket #10481 is fixed:

```
sage: var(x)
x
sage: R.<a>=ZZ[x].quo(x^2)
sage: R.fraction_field()
Traceback (most recent call last):
...
NotImplementedError
sage: R.is_integral_domain()
Traceback (most recent call last):
...
NotImplementedError
```

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 $\mathbf{Q}$ or $\mathbf{F}_p$.

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 NotImplementedError if not known.

EXAMPLES:

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

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

EXAMPLES:

```
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,)

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.

TESTS:

The following example returns a `NotImplementedError` since the generic ring class `__call__` function returns a `NotImplementedError`. Note that `sage.rings.ring.Ring.random_element` performs a call in the generic ring class by a random integer:

```
sage: R = sage.rings.ring.Ring(ZZ); R
<type 'sage.rings.ring.Ring'>
sage: R.random_element()
Traceback (most recent call last):
...
NotImplementedError
```

unit_ideal()

Return the unit ideal of this ring.

EXAMPLES:

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

zero()

Return the zero element of this ring (cached).

EXAMPLES:

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

The result is cached:

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

zero_ideal()

Return the zero ideal of this ring (cached).

EXAMPLES:

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

The result is cached:

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

TESTS:

Make sure that trac ticket #13644 is fixed:

```
sage: K = Qp(3)
sage: R.<a> = K[]
sage: L.<a> = K.extension(a^2-3)
sage: L.ideal(a)
Principal ideal (1 + O(a^40)) of Eisenstein Extension of 3-adic Field with capped relative p
```

zeta(*n=2*, *all=False*)

Return an `n`-th root of unity in `self` if there is one, or raise an `ArithmeticError` otherwise.

INPUT:

- `n` – positive integer
- `all` – bool, default: False. If True, return a list of all n-th roots of 1.

OUTPUT:

Element of `self` of finite order

EXAMPLES:

```
sage: QQ.zeta()
-1
sage: QQ.zeta(1)
1
sage: CyclotomicField(6).zeta()
zeta6
sage: CyclotomicField(3).zeta()
zeta3
sage: CyclotomicField(3).zeta().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
sage: Zp(7, prec=8).zeta()
3 + 4*7 + 6*7^2 + 3*7^3 + 2*7^5 + 6*7^6 + 2*7^7 + O(7^8)
```

zeta_order()

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

EXAMPLES:

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

CHAPTER

TWO

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.ideal([a,b,...])`. The case of non-commutative rings is implemented in `noncommutative_ideals`.

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

`sage.rings.ideal.`**`Cyclic`**(*R*, *n=None*, *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
```

`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

sage: ideal(x^2-2*x+1, x^2-1)
Ideal (x^2 - 2*x + 1, x^2 - 1) of Univariate Polynomial Ring in x over Integer Ring
sage: ideal([x^2-2*x+1, x^2-1])
Ideal (x^2 - 2*x + 1, x^2 - 1) of Univariate Polynomial Ring in x over Integer Ring
sage: l = [x^2-2*x+1, x^2-1]
sage: ideal(f^2 for f in l)
Ideal (x^4 - 4*x^3 + 6*x^2 - 4*x + 1, x^4 - 2*x^2 + 1) of
Univariate Polynomial Ring in x over Integer Ring
```

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

```
sage: R.<t> = ZZ['t']
sage: i = ideal(1,t,t^2)
sage: i
Ideal (1, t, t^2) of Univariate Polynomial Ring in t over Integer Ring
sage: ideal(1/2,t,t^2)
Principal ideal (1) of Univariate Polynomial Ring in t over Rational Field
```

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

```
sage: Ideal(3, 5)
Principal ideal (1) of Integer Ring
sage: Ideal(ZZ, 3, 5)
Principal ideal (1) of Integer Ring
sage: Ideal(2, 4, 6)
Principal ideal (2) of Integer Ring
```

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

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

sage: I = Ideal()
Traceback (most recent call last):
...
ValueError: need at least one argument
```

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

```
sage: R.<x> = GF(5)[]
sage: I = R*(x^2+3)
sage: type(I)
<class 'sage.rings.polynomial.ideal.Ideal_1poly_field'>
```

You can also pass in a specific ideal type:

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

TESTS:

```
sage: R.<x> = ZZ[]
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: I == loads(dumps(I))
True

sage: I = Ideal(R, [4 + 3*x + x^2, 1 + x^2])
sage: I == loads(dumps(I))
True

sage: I = Ideal((4 + 3*x + x^2, 1 + x^2))
sage: I == loads(dumps(I))
True
```

This shows that the issue at trac ticket #5477 is fixed:

```
sage: R.<x> = QQ[]
sage: I = R.ideal([x + x^2])
sage: J = R.ideal([2*x + 2*x^2])
sage: J
Principal ideal (x^2 + x) of Univariate Polynomial Ring in x over Rational Field
sage: S = R.quotient_ring(I)
sage: U = R.quotient_ring(J)
sage: I == J
True
sage: S == U
True
```

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

Bases: `sage.rings.ideal.Ideal_generic`

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
sage: psi(J)
Principal ideal (x - 1.00000000000000) of Univariate Polynomial Ring in x over Complex Field
sage: J.apply_morphism(psi)
Principal ideal (x - 1.00000000000000) of Univariate Polynomial Ring in x over Complex Field
```

```
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
```

TESTS:

```
sage: K.<a> = NumberField(x^2 + 1)
sage: A = K.ideal(a)
sage: taus = K.embeddings(K)
sage: A.apply_morphism(taus[0]) # identity
Fractional ideal (a)
sage: A.apply_morphism(taus[1]) # complex conjugation
Fractional ideal (-a)
sage: A.apply_morphism(taus[0]) == A.apply_morphism(taus[1])
True
```

```
sage: K.<a> = NumberField(x^2 + 5)
sage: B = K.ideal([2, a + 1]); B
Fractional ideal (2, a + 1)
sage: taus = K.embeddings(K)
sage: B.apply_morphism(taus[0]) # identity
Fractional ideal (2, a + 1)
```

Since 2 is totally ramified, complex conjugation fixes it:

```
sage: B.apply_morphism(taus[1]) # complex conjugation
Fractional ideal (2, a + 1)
sage: taus[1](B)
Fractional ideal (2, a + 1)
```

associated_primes()

Return the list of associated prime ideals of this ideal.

EXAMPLES:

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

base_ring()

Returns the base ring of this ideal.

EXAMPLES:

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

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

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

And p-adic numbers:

```
sage: R = Zp(7, prec=10); R
7-adic Ring with capped relative precision 10
sage: I = 7*R; I
Principal ideal (7 + O(7^11)) of 7-adic Ring with capped relative precision 10
sage: I.base_ring()
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: I = ZZ.ideal(7)
sage: J = ZZ[x].ideal(7,x)
sage: K = ZZ[x].ideal(7)
sage: I.category()
Category of ring ideals in Integer Ring
sage: J.category()
Category of ring ideals in Univariate Polynomial Ring in x
over Integer Ring
sage: K.category()
Category of ring ideals in Univariate Polynomial Ring in x
over Integer Ring
```

embedded_primes()

Return the list of embedded primes of this ideal.

EXAMPLES:

```
sage: 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.

EXAMPLE:

```
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.

EXAMPLE:

```
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 $\mathbf{Z}$ have the method `gens_reduced()`, just like ideals of number fields.

EXAMPLES:

```
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:

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

is_primary(*P=None*)

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

Recall that an ideal I is primary if and only if I has a unique associated prime (see page 52 in [AtiMac]). 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:

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

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

Some examples from the Macaulay2 documentation:

```
sage: R.<x, y, z> = GF(101)[]
sage: I = R.ideal([y^6])
sage: I.is_primary()
True
sage: I.is_primary(R.ideal([y]))
True
sage: I = R.ideal([x^4, y^7])
sage: I.is_primary()
True
sage: I = R.ideal([x*y, y^2])
sage: I.is_primary()
False
```

REFERENCES:

is_prime()

Return `True` if this ideal is prime.

EXAMPLES:

```
sage: R.<x, y> = QQ[]
sage: I = R.ideal([x, y])
sage: I.is_prime()         # a maximal ideal
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()
False
sage: S.ideal(2).is_prime()
True
sage: S.ideal(4).is_prime()
False
```

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

```
sage: R.<x> = ZZ[]
sage: I = R.ideal(7)
sage: I.is_prime()         # when implemented, should be True
Traceback (most recent call last):
...
```

```
NotImplementedError
```

Note: For general rings, uses the list of associated primes.

is_principal()

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

Todo

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

EXAMPLES:

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

is_trivial()

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

TESTS:

```
sage: I = ZZ.ideal(5)
sage: I.is_trivial()
False

sage: I = ZZ['x'].ideal(-1)
sage: I.is_trivial()
True

sage: I = ZZ['x'].ideal(ZZ['x'].gen()^2)
sage: I.is_trivial()
False

sage: I = QQ['x', 'y'].ideal(-5)
sage: I.is_trivial()
True

sage: I = CC['x'].ideal(0)
sage: I.is_trivial()
True
```

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.

EXAMPLE:

```
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 s
```

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
```

reduce(f)

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

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

EXAMPLES:

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

ring()

Returns the ring containing this ideal.

EXAMPLES:

```
sage: R = ZZ
sage: I = 3*R; I
```

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

Note that `self.ring()` is different from `self.base_ring()`

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

Another example using polynomial rings:

```
sage: R = PolynomialRing(QQ, 'x'); x = R.gen()
sage: I = R.ideal(x^2 - 3)
sage: I.ring()
Univariate Polynomial Ring in x over Rational Field
sage: Rbar = R.quotient(I, names='a')
sage: S = PolynomialRing(Rbar, 'y'); y = Rbar.gen(); S
Univariate Polynomial Ring in y over Univariate Quotient Polynomial Ring in a over Rational
sage: J = S.ideal(y^2 + 1)
sage: J.ring()
Univariate Polynomial Ring in y over Univariate Quotient Polynomial Ring in a over Rational
```

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

Bases: `sage.rings.ideal.Ideal_principal`

An ideal of a principal ideal domain.

See `Ideal()`.

`gcd`(*other*)

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

EXAMPLES:

An example in the principal ideal domain **Z**:

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

TESTS:

We cannot take the gcd of a principal ideal with a non-principal ideal as well: (`gcd(I,J)` should be (7))

```
sage: I = ZZ.ideal(7)
sage: J = ZZ[x].ideal(7,x)
sage: I.gcd(J)
Traceback (most recent call last):
```

```
...
NotImplementedError
sage: J.gcd(I)
Traceback (most recent call last):
...
AttributeError: 'Ideal_generic' object has no attribute 'gcd'
```

Note:

```
sage: type(I)
<class 'sage.rings.ideal.Ideal_pid'>
sage: type(J)
<class 'sage.rings.ideal.Ideal_generic'>
```

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: 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()

Return `True` if the ideal is prime.

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

EXAMPLES:

```
sage: ZZ.ideal(2).is_prime()
True
sage: ZZ.ideal(-2).is_prime()
True
sage: ZZ.ideal(4).is_prime()
False
sage: ZZ.ideal(0).is_prime()
True
sage: R.<x>=QQ[]
sage: P=R.ideal(x^2+1); P
Principal ideal (x^2 + 1) of Univariate Polynomial Ring in x over Rational Field
sage: P.is_prime()
True
```

In fields, only the zero ideal is prime:

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

reduce (f)

Return the reduction of f modulo `self`.

EXAMPLES:

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

residue_field ()

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

EXAMPLES:

```
sage: P = ZZ.ideal(61); P
Principal ideal (61) of Integer Ring
sage: F = P.residue_field(); F
Residue field of Integers modulo 61
sage: pi = F.reduction_map(); pi
Partially defined reduction map:
  From: Rational Field
  To:   Residue field of Integers modulo 61
sage: pi(123/234)
6
sage: pi(1/61)
Traceback (most recent call last):
...
ZeroDivisionError: Cannot reduce rational 1/61 modulo 61: it has negative valuation
sage: lift = F.lift_map(); lift
Lifting map:
  From: Residue field of Integers modulo 61
  To:   Integer Ring
sage: lift(F(12345/67890))
33
sage: (12345/67890) % 61
33
```

TESTS:

```
sage: ZZ.ideal(96).residue_field()
Traceback (most recent call last):
...
ValueError: The ideal (Principal ideal (96) of Integer Ring) is not prime

sage: R.<x>=QQ[]
sage: I=R.ideal(x^2+1)
sage: I.is_prime()
True
sage: I.residue_field()
Traceback (most recent call last):
...
TypeError: residue fields only supported for polynomial rings over finite fields.
```

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 = ZZ[x]
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:

```
sage: R = ZZ[x]
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:

```
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,1)
sage: J = sage.rings.ideal.Katsura(Q,1); J
Ideal (x - 1) of Multivariate Polynomial Ring in x over Rational Field
```

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

Return True if object is an ideal of a ring.

EXAMPLES:

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

```
sage: from sage.rings.ideal import is_Ideal
sage: R = ZZ
sage: is_Ideal(R)
False
sage: 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()
sage: I = R.ideal(x^2 + 1); I
Principal ideal (x^2 + 1) of Univariate Polynomial Ring in x over Rational Field
sage: is_Ideal(I)
True
sage: is_Ideal((x^2 + 1)*R)
True
```

CHAPTER

THREE

MONOID OF IDEALS IN A COMMUTATIVE RING

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

Return the monoid of ideals in the ring R.

EXAMPLE:

```
sage: R = QQ['x']
sage: sage.rings.ideal_monoid.IdealMonoid(R)
Monoid 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.

TESTS:

```
sage: R = QQ['x']
sage: M = sage.rings.ideal_monoid.IdealMonoid(R)
sage: TestSuite(M).run()
  Failure in _test_category:
...
The following tests failed: _test_elements
```

(The "_test_category" test fails but I haven't the foggiest idea why.)

Element

alias of Ideal_generic

ring()

Return the ring of which this is the ideal monoid.

EXAMPLE:

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

CHAPTER

FOUR

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.

TESTS:

```
sage: A = SteenrodAlgebra(2)
sage: IL = A*[A.1+A.2,A.1^2]; IL
Left Ideal (Sq(2) + Sq(4), Sq(1,1)) of mod 2 Steenrod algebra, milnor basis
sage: TestSuite(IL).run(skip=['_test_category'],verbose=True)
running ._test_eq() . . . pass
running ._test_not_implemented_methods() . . . pass
running ._test_pickling() . . . pass
```

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],

  [0 0]
  [1 0]
)
 of Full MatrixSpace of 2 by 2 dense matrices over Rational Field
sage: [MS.1,MS.2]*MS
Right Ideal
(
  [0 1]
  [0 0],

  [0 0]
  [1 0]
)
 of Full MatrixSpace of 2 by 2 dense matrices over Rational Field
sage: MS*[MS.1,MS.2]*MS
Twosided Ideal
(
  [0 1]
  [0 0],

  [0 0]
  [1 0]
)
 of Full MatrixSpace of 2 by 2 dense matrices over Rational Field
```

side()

Return a string that describes the sidedness of this ideal.

EXAMPLES:

```
sage: 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'
```

CHAPTER

FIVE

HOMOMORPHISMS OF RINGS

We give a large number of examples of ring homomorphisms.

EXAMPLE:

Natural inclusion $\mathbf{Z} \hookrightarrow \mathbf{Q}$:

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

There is no homomorphism in the other direction:

```
sage: H = Hom(QQ, ZZ)
sage: H([1])
Traceback (most recent call last):
...
TypeError: images do not define a valid homomorphism
```

EXAMPLES:

Reduction to finite field:

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

Map from single variable polynomial ring:

```
sage: R.<x> = ZZ[]
sage: phi = R.hom([2], GF(5))
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([RR(1)]); f
Ring endomorphism of Real Field with 53 bits of precision
  Defn: 1.00000000000000 |--> 1.00000000000000
sage: f(2.5)
2.50000000000000
sage: f = RR.hom( [2.0] )
Traceback (most recent call last):
...
TypeError: images do not define a valid homomorphism
```

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 20
```

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:

```
sage: i = RR.hom([CC(1)]); i
Ring morphism:
  From: Real Field with 53 bits of precision
  To:   Complex Field with 53 bits of precision
  Defn: 1.00000000000000 |--> 1.00000000000000
sage: i(RR('3.1'))
3.10000000000000
```

A map from a multivariate polynomial ring to itself:

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

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

```
sage: R.<x,y> = PolynomialRing(QQ)
sage: S.<a,b> = quo(R, ideal(1 + y^2))
sage: phi = S.hom([a^2, -b])
```

```
sage: phi
Ring endomorphism of Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the idea
  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 `GF(2)` into `GF(4,'a')`:

```
sage: k = GF(2)
sage: i = k.hom(GF(4, 'a'))
sage: i
Ring Coercion morphism:
  From: Finite Field of size 2
  To:   Finite Field in a of size 2^2
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 `GF(2)`:

```
sage: pi = ZZ.hom(k)
sage: pi
Ring Coercion 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:   Ring Coercion morphism:
```

```
          From: Integer Ring
          To:   Finite Field of size 2
        then
          Ring Coercion morphism:
          From: Finite Field of size 2
          To:   Finite Field in a of size 2^2
sage: a = f(5); a
1
sage: a.parent()
Finite Field in a of size 2^2
```

Inclusion from **Q** to the 3-adic field:

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

An automorphism of a quotient of a univariate polynomial ring:

```
sage: R.<x> = PolynomialRing(QQ)
sage: S.<sqrt2> = R.quo(x^2-2)
sage: sqrt2^2
2
sage: (3+sqrt2)^10
993054*sqrt2 + 1404491
sage: c = S.hom([-sqrt2])
sage: c(1+sqrt2)
-sqrt2 + 1
```

Note that Sage verifies that the morphism is valid:

```
sage: (1 - sqrt2)^2
-2*sqrt2 + 3
sage: c = S.hom([1-sqrt2])     # this is not valid
Traceback (most recent call last):
...
TypeError: images do not define a valid homomorphism
```

Endomorphism of power series ring:

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

Homomorphism of Laurent series ring:

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

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

```
sage: R.hom([1/t])
Traceback (most recent call last):
...
TypeError: images do not define a valid homomorphism
sage: R.hom([1])
Traceback (most recent call last):
...
TypeError: images do not define a valid homomorphism
```

Complex conjugation on cyclotomic fields:

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

Embedding a number field into the reals:

```
sage: 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:

```
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
```

TESTS:

```
sage: H = Hom(ZZ, QQ)
sage: H == loads(dumps(H))
True
```

```
sage: K.<zeta7> = CyclotomicField(7)
sage: c = K.hom([1/zeta7])
sage: c == loads(dumps(c))
True
```

```
sage: R.<t> = PowerSeriesRing(GF(5))
sage: f = R.hom([t^5])
sage: f == loads(dumps(f))
True
```

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 endormorphism 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(ZZ)
sage: f.inverse_image(ZZ.ideal(2))
Traceback (most recent call last):
...
NotImplementedError
```

is_injective()

Return whether or not this morphism is injective, or raise a NotImplementedError.

EXAMPLES:

Note that currently this is not implemented in most interesting cases:

```
sage: f = ZZ.hom(QQ)
sage: f.is_injective()
Traceback (most recent call last):
...
NotImplementedError
```

is_zero()

Return True if this is the zero map and False otherwise.

A *ring* homomorphism is considered to be 0 if and only if it sends the 1 element of the domain to the 0 element of the codomain. Since rings in Sage all have a 1 element, the zero homomorphism is only to a ring of order 1, where `1 == 0`, e.g., the ring `Integers(1)`.

EXAMPLES:

First an example of a map that is obviously nonzero:

```
sage: h = Hom(ZZ, QQ)
sage: f = h.natural_map()
sage: f.is_zero()
False
```

Next we make the zero ring as $\mathbf{Z}/1\mathbf{Z}$:

```
sage: R = Integers(1)
sage: R
Ring of integers modulo 1
```

```
sage: h = Hom(ZZ, R)
sage: f = h.natural_map()
sage: f.is_zero()
True
```

Finally we check an example in characteristic 2:

```
sage: h = Hom(ZZ, GF(2))
sage: f = h.natural_map()
sage: f.is_zero()
False
```

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 F
```

class `sage.rings.morphism.`**RingHomomorphism_coercion**

Bases: `sage.rings.morphism.RingHomomorphism`

Initialize `self`.

INPUT:

- `parent` – ring homset
- `check` – bool (default: `True`)

EXAMPLES:

```
sage: f = ZZ.hom(QQ); f                    # indirect doctest
Ring Coercion morphism:
  From: Integer Ring
  To:   Rational Field

sage: f == loads(dumps(f))
True
```

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 +
  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
Ring morphism:
  From: Univariate Polynomial Ring in t over Multivariate Polynomial Ring in x, y over Rational
  To:   Univariate Polynomial Ring in t over Univariate Polynomial Ring in z over Rational Field
  Defn: Induced from base ring by
        Ring morphism:
          From: Multivariate Polynomial Ring in x, y over Rational Field
          To:   Univariate Polynomial Ring in z over Rational Field
          Defn: x |--> 2*z
                y |--> 3*z
sage: p = (x - 4*y + 1/13)*t^2 + (1/2*x^2 - 1/3*y^2)*t + 2*y^2 + x
sage: Pf(p)
(-10*z + 1/13)*t^2 - z^2*t + 18*z^2 + 2*z
```

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

```
sage: MR = MatrixSpace(R,2,2)
sage: MS = MatrixSpace(S,2,2)
sage: M = MR([x^2 + 1/7*x*y - y^2, - 1/2*y^2 + 2*y + 1/6, 4*x^2 - 14*x, 1/2*y^2 + 13/4*x - 2/11*
sage: Mf = MR.hom(f,MS)
sage: Mf
Ring morphism:
  From: Full MatrixSpace of 2 by 2 dense matrices over Multivariate Polynomial Ring in x, y over
  To:   Full MatrixSpace of 2 by 2 dense matrices over Univariate Polynomial Ring in z over Rati
  Defn: Induced from base ring by
        Ring morphism:
          From: Multivariate Polynomial Ring in x, y over Rational Field
          To:   Univariate Polynomial Ring in z over Rational Field
          Defn: x |--> 2*z
                y |--> 3*z
sage: Mf(M)
[           -29/7*z^2 -9/2*z^2 + 6*z + 1/6]
[       16*z^2 - 28*z   9/2*z^2 + 131/22*z]
```

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

```
sage: MPR = MatrixSpace(PR, 2)
sage: MPS = MatrixSpace(PS, 2)
sage: M = MPR([(- x + y)*t^2 + 58*t - 3*x^2 + x*y, (- 1/7*x*y - 1/40*x)*t^2 + (5*x^2 + y^2)*t +
sage: MPf = MPR.hom(f,MPS); MPf
Ring morphism:
  From: Full MatrixSpace of 2 by 2 dense matrices over Univariate Polynomial Ring in t over Mult
  To:   Full MatrixSpace of 2 by 2 dense matrices over Univariate Polynomial Ring in t over Univ
  Defn: Induced from base ring by
        Ring morphism:
          From: Univariate Polynomial Ring in t over Multivariate Polynomial Ring in x, y over R
          To:   Univariate Polynomial Ring in t over Univariate Polynomial Ring in z over Ration
          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 + 6*z]
[    (-z + 1)*t^2 + 11*z^2 + 15/2*z + 1/4                           (20*z + 1)*t^2]
```

is_identity()

Return `True` if this morphism is the identity morphism.

EXAMPLES:

```
sage: K.<z> = GF(4)
sage: phi = End(K)([z^2])
sage: R.<t> = K[]
sage: psi = End(R)(phi)
sage: psi.is_identity()
False
```

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 Hom(R,S)
- phi – a ring homomorphism C --> S, where C is the domain of R.cover()

OUTPUT: a ring homomorphism

The domain R is a quotient object $C \to R$, and R.cover() is the ring homomorphism $\varphi : C \to R$. The condition on the elements im_gens of S is that they define a homomorphism $C \to S$ such that each generator of the kernel of φ 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
  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 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:

```
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
  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:

```
sage: R.<x,y> = QQ[]
sage: f = R.hom([x,x+y])
sage: f.im_gens()
[x, x + y]
```

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

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

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 $R \to S$ sending x to x.lift().

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: S.<xbar,ybar> = R.quo( (x^2 + y^2, y) )
sage: S.lift()
Set-theoretic ring morphism:
  From: Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x^2 +
  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.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)
True
sage: sage.rings.morphism.is_RingHomomorphism(2/3)
False
```

CHAPTER

SIX

SPACE OF HOMOMORPHISMS BETWEEN TWO RINGS

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

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

For more on homsets, see Hom().

EXAMPLES:

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

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

Bases: 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 implementation work is done).

Currently this checks if the domains and the codomains are equal.

EXAMPLES:

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

natural_map()

Returns the natural map from the domain to the codomain.

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

EXAMPLES:

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

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 Rational Field by the
  Defn: a |--> b
        b |--> a
sage: phi(a)
b
sage: phi(b)
a
```

TESTS:

We test pickling of a homset from a quotient.

```
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = R.quotient(x^2 + y^2)
sage: H = S.Hom(R)
sage: H == loads(dumps(H))
True
```

We test pickling of actual homomorphisms in a quotient:

```
sage: phi = S.hom([b,a])
sage: phi == loads(dumps(phi))
True
```

`sage.rings.homset.is_RingHomset`(*H*)

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

EXAMPLES:

```
sage: from sage.rings.homset import is_RingHomset as is_RH
sage: is_RH(Hom(ZZ, QQ))
True
sage: is_RH(ZZ)
False
sage: is_RH(Hom(RR, CC))
True
sage: is_RH(Hom(FreeModule(ZZ,1), FreeModule(QQ,1)))
False
```

CHAPTER SEVEN

INFINITY RINGS

The unsigned infinity "ring" is the set of two elements

1. infinity
2. A number less than infinity

The rules for arithmetic are that the unsigned infinity ring does not canonically coerce to any other ring, and all other rings canonically coerce to the unsigned infinity ring, sending all elements to the single element "a number less than infinity" of the unsigned infinity ring. Arithmetic and comparisons then take place in the unsigned infinity ring, where all arithmetic operations that are well-defined are defined.

The infinity "ring" is the set of five elements

1. plus infinity
2. a positive finite element
3. zero
4. a negative finite element
5. negative infinity

The infinity ring coerces to the unsigned infinity ring, sending the infinite elements to infinity and the non-infinite elements to "a number less than infinity." Any ordered ring coerces to the infinity ring in the obvious way.

Note: The shorthand `oo` is predefined in Sage to be the same as `+Infinity` in the infinity ring. It is considered equal to, but not the same as `Infinity` in the `UnsignedInfinityRing`.

EXAMPLES:

We fetch the unsigned infinity ring and create some elements:

```
sage: P = UnsignedInfinityRing; P
The Unsigned Infinity Ring
sage: P(5)
A number less than infinity
sage: P.ngens()
1
sage: unsigned_oo = P.0; unsigned_oo
Infinity
```

We compare finite numbers with infinity:

```
sage: 5 < unsigned_oo
True
sage: 5 > unsigned_oo
False
```

```
sage: unsigned_oo < 5
False
sage: unsigned_oo > 5
True
```

Demonstrating the shorthand `oo` versus `Infinity`:

```
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: unsigned oo times smaller number not defined
```

What happened above is that 0 is canonically coerced to "a number less than infinity" in the unsigned infinity ring, and the quotient is then 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 `oo` is predefined as a shorthand for `+Infinity`:

```
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 / oo` and `1 / -oo` return the integer 0 instead of the infinity ring Zero so that arithmetic of the following type works:

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

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

```
sage: oo - oo
Traceback (most recent call last):
...
SignError: cannot add infinity to minus infinity
sage: 0 * oo
Traceback (most recent call last):
...
SignError: cannot multiply infinity by zero
sage: P(2) + P(-3)
Traceback (most recent call last):
...
SignError: cannot add positive finite value to negative finite value
```

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

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

TESTS:

```
sage: P = InfinityRing
sage: P == loads(dumps(P))
True
```

```
sage: P(2) == loads(dumps(P(2)))
True
```

The following is assumed in a lot of code (i.e., "is" is used for testing whether something is infinity), so make sure it is satisfied:

```
sage: loads(dumps(infinity)) is infinity
True
```

We check that trac ticket #17990 is fixed:

```
sage: m = Matrix([Infinity])
sage: m.rows()
[(+Infinity)]
```

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`.

TESTS:

```
sage: sage.rings.infinity.FiniteNumber(InfinityRing, 1)
A positive finite number
sage: sage.rings.infinity.FiniteNumber(InfinityRing, -1)
A negative finite number
sage: sage.rings.infinity.FiniteNumber(InfinityRing, 0)
Zero
```

`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
```

class sage.rings.infinity.**InfinityRing_class**

Bases: sage.rings.infinity._uniq, sage.rings.ring.Ring

Initialize self.

TEST:

```
sage: sage.rings.infinity.InfinityRing_class() is sage.rings.infinity.InfinityRing_class() is In
True
```

fraction_field()

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

TEST:

```
sage: InfinityRing.fraction_field()
Traceback (most recent call last):
...
TypeError: infinity 'ring' has no fraction field
```

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.MinusInfinityElement

Initialize self.

TESTS:

```
sage: sage.rings.infinity.MinusInfinity() is sage.rings.infinity.MinusInfinity() is -oo
True
```

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.PlusInfinityElement

Initialize self.

TESTS:

```
sage: sage.rings.infinity.PlusInfinity() is sage.rings.infinity.PlusInfinity() is oo
True
```

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.

TESTS:

```
sage: sage.rings.infinity.UnsignedInfinity() is sage.rings.infinity.UnsignedInfinity() is unsign
True
```

class sage.rings.infinity.**UnsignedInfinityRing_class**

Bases: sage.rings.infinity._uniq, sage.rings.ring.Ring

Initialize self.

TESTS:

```
sage: sage.rings.infinity.UnsignedInfinityRing_class() is sage.rings.infinity.UnsignedInfinityRi
True
```

fraction_field()

The unsigned infinity ring isn't an integral domain.

EXAMPLES:

```
sage: UnsignedInfinityRing.fraction_field()
Traceback (most recent call last):
...
TypeError: infinity 'ring' has no fraction field
```

gen(*n=0*)

The "generator" of self is the infinity object.

EXAMPLES:

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

gens()

The "generator" of self is the infinity object.

EXAMPLES:

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

less_than_infinity()

This is the element that represents a finite value.

EXAMPLES:

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

ngens()

The unsigned infinity ring has one "generator."

EXAMPLES:

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

sage.rings.infinity.**is_Infinite**(*x*)

This is a type check for infinity elements.

EXAMPLES:

```
sage: sage.rings.infinity.is_Infinite(oo)
True
sage: sage.rings.infinity.is_Infinite(-oo)
True
sage: sage.rings.infinity.is_Infinite(unsigned_infinity)
True
sage: sage.rings.infinity.is_Infinite(3)
False
sage: sage.rings.infinity.is_Infinite(RR(infinity))
False
sage: sage.rings.infinity.is_Infinite(ZZ)
False
```

sage.rings.infinity.**test_comparison**(*ring*)

Check comparison with infinity

INPUT:

- `ring` – a sub-ring of the real numbers

OUTPUT:

Various attempts are made to generate elements of `ring`. An assertion is triggered if one of these elements does not compare correctly with plus/minus infinity.

EXAMPLES:

```
sage: from sage.rings.infinity import test_comparison
sage: rings = [ZZ, QQ, RR, RealField(200), RDF, RLF, 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:

```
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:

```
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:

```
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)
```

CHAPTER

EIGHT

FRACTION FIELD OF INTEGRAL DOMAINS

AUTHORS:

- William Stein (with input from David Joyner, David Kohel, and Joe Wetherell)
- Burcin Erocal

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)
```

TESTS:

```
sage: F = FractionField(IntegerRing())
sage: F == loads(dumps(F))
True
```

```
sage: F = FractionField(PolynomialRing(RationalField(),'x'))
sage: F == loads(dumps(F))
True
```

```
sage: F = FractionField(PolynomialRing(IntegerRing(),'x'))
sage: F == loads(dumps(F))
True
```

```
sage: F = FractionField(PolynomialRing(RationalField(),2,'x'))
sage: F == loads(dumps(F))
True
```

`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
sage: FractionField(PolynomialRing(IntegerRing(),'x'))
Fraction Field of Univariate Polynomial Ring in x over Integer Ring
sage: FractionField(PolynomialRing(RationalField(),2,'x'))
Fraction Field of Multivariate Polynomial Ring in x0, x1 over Rational Field
```

Dividing elements often implicitly creates elements of the fraction field:

```
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:

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

class sage.rings.fraction_field.**FractionField_1poly_field**(*R*, *element_class=<class 'sage.rings.fraction_field_element.FractionFieldEle*

Bases: sage.rings.fraction_field.FractionField_generic

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

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

class_number()

Here for compatibility with number fields and function fields.

EXAMPLES:

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

maximal_order()

Returns the maximal order in this fraction field.

EXAMPLES:

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

ring_of_integers()

Returns 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
```

`construction`()

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 ove
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.

EXAMPLE:

```
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 ove
sage: R.ngens()
10
```

random_element(**args, **kwds*)

Returns 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
```

`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
```

CHAPTER

NINE

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 [Ho72]

REFERENCES:

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
```

TESTS:

Test if trac ticket #5451 is fixed:

```
sage: A = FiniteField(9,'theta')['t']
sage: K.<t> = FractionField(A)
sage: f= 2/(t^2+2*t); g =t^9/(t^18 + t^10 + t^2);f+g
(2*t^15 + 2*t^14 + 2*t^13 + 2*t^12 + 2*t^11 + 2*t^10 + 2*t^9 + t^7 + t^6 + t^5 + t^4 + t^3 + t^2
```

Test if trac ticket #8671 is fixed:

```
sage: P.<n> = QQ[]
sage: F = P.fraction_field()
sage: P.one()//F.one()
1
sage: F.one().quo_rem(F.one())
(1, 0)
```

`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*)

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

INPUT:

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

OUTPUT:

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

EXAMPLES:

```
sage: R.<t> = QQ[]
sage: (1/t).is_square()
False
sage: (1/t^6).is_square()
True
sage: ((1+t)^4/t^6).is_square()
True
sage: (4*(1+t)^4/t^6).is_square()
True
sage: (2*(1+t)^4/t^6).is_square()
False
sage: ((1+t)/t^6).is_square()
False

sage: (4*(1+t)^4/t^6).is_square(root=True)
(True, (2*t^2 + 4*t + 2)/t^3)
sage: (2*(1+t)^4/t^6).is_square(root=True)
(False, None)

sage: R.<x> = QQ[]
sage: a = 2*(x+1)^2 / (2*(x-1)^2); a
(2*x^2 + 4*x + 2)/(2*x^2 - 4*x + 2)
sage: a.numerator().is_square()
False
sage: a.is_square()
True
```

```
sage: (0/x).is_square()
True
```

is_zero()

Return `True` if this element is equal to zero.

EXAMPLES:

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

numerator()

Return the numerator of `self`.

EXAMPLES:

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

reduce()

Divides out the gcd of the numerator and denominator.

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

EXAMPLES:

```
sage: R.<x> = RealField(10)[]
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
```

valuation(*v=None*)

Return the valuation of `self`, assuming that the numerator and denominator have valuation functions defined on them.

EXAMPLES:

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

class `sage.rings.fraction_field_element.`**FractionFieldElement_1poly_field**

Bases: `sage.rings.fraction_field_element.FractionFieldElement`

A fraction field element where the parent is the fraction field of a univariate polynomial ring.

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

is_integral()

Returns whether this element is actually a polynomial.

EXAMPLES:

```
sage: R.<t> = QQ[]
sage: elt = (t^2 + t - 2) / (t + 2); elt # == (t + 2)*(t - 1)/(t + 2)
t - 1
sage: elt.is_integral()
True
sage: elt = (t^2 - t) / (t+2); elt # == t*(t - 1)/(t + 2)
(t^2 - t)/(t + 2)
sage: elt.is_integral()
False
```

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 +
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*)

Returns whether or not x is a :class'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__denomi
(x + y)/(x - y)
```

CHAPTER

TEN

UNIVARIATE RATIONAL FUNCTIONS OVER PRIME FIELDS

class `sage.rings.fraction_field_FpT.`**`FpT`**(*R*, *names=None*)

Bases: `sage.rings.fraction_field.FractionField_1poly_field`

This class represents the fraction field GF(p)(T) for $2 < p < 2^16$.

EXAMPLES:

```
sage: R.<T> = GF(71)[]
sage: K = FractionField(R); K
Fraction Field of Univariate Polynomial Ring in T over Finite Field of size 71
sage: 1-1/T
(T + 70)/T
sage: parent(1-1/T) is K
True
```

`iter`(*bound=None*, *start=None*)

EXAMPLES:

```
sage: from sage.rings.fraction_field_FpT import *
sage: R.<t> = FpT(GF(5)['t'])
sage: list(R.iter(2))[350:355]
[(t^2 + t + 1)/(t + 2),
 (t^2 + t + 2)/(t + 2),
 (t^2 + t + 4)/(t + 2),
 (t^2 + 2*t + 1)/(t + 2),
 (t^2 + 2*t + 2)/(t + 2)]
```

class `sage.rings.fraction_field_FpT.`**`FpTElement`**

Bases: `sage.structure.element.RingElement`

An element of an FpT fraction field.

`denom`()

Returns the denominator of this element, as an element of the polynomial ring.

EXAMPLES:

```
sage: K = GF(11)['t'].fraction_field()
sage: t = K.gen(0); a = (t + 1/t)^3 - 1
sage: a.denom()
t^3
```

`denominator`()

Returns the denominator of this element, as an element of the polynomial ring.

EXAMPLES:

```
sage: K = GF(11)['t'].fraction_field()
sage: t = K.gen(0); a = (t + 1/t)^3 - 1
sage: a.denominator()
t^3
```

factor()

EXAMPLES:

```
sage: K = Frac(GF(5)['t'])
sage: t = K.gen()
sage: f = 2 * (t+1) * (t^2+t+1)^2 / (t-1)
sage: factor(f)
(2) * (t + 4)^-1 * (t + 1) * (t^2 + t + 1)^2
```

is_square()

Returns True if this element is the square of another element of the fraction field.

EXAMPLES:

```
sage: K = GF(13)['t'].fraction_field(); t = K.gen()
sage: t.is_square()
False
sage: (1/t^2).is_square()
True
sage: K(0).is_square()
True
```

next()

This function iterates through all polynomials, returning the "next" polynomial after this one.

The strategy is as follows:

- We always leave the denominator monic.
- We progress through the elements with both numerator and denominator monic, and with the denominator less than the numerator. For each such, we output all the scalar multiples of it, then all of the scalar multiples of its inverse.
- So if the leading coefficient of the numerator is less than p-1, we scale the numerator to increase it by 1.
- Otherwise, we consider the multiple with numerator and denominator monic.
  - If the numerator is less than the denominator (lexicographically), we return the inverse of that element.
  - If the numerator is greater than the denominator, we invert, and then increase the numerator (remaining monic) until we either get something relatively prime to the new denominator, or we reach the new denominator. In this case, we increase the denominator and set the numerator to 1.

EXAMPLES:

```
sage: from sage.rings.fraction_field_FpT import *
sage: R.<t> = FpT(GF(3)['t'])
sage: a = R(0)
sage: for _ in range(30):
...       a = a.next()
...       print a
...
1
2
1/t
```

```
2/t
t
2*t
1/(t + 1)
2/(t + 1)
t + 1
2*t + 2
t/(t + 1)
2*t/(t + 1)
(t + 1)/t
(2*t + 2)/t
1/(t + 2)
2/(t + 2)
t + 2
2*t + 1
t/(t + 2)
2*t/(t + 2)
(t + 2)/t
(2*t + 1)/t
(t + 1)/(t + 2)
(2*t + 2)/(t + 2)
(t + 2)/(t + 1)
(2*t + 1)/(t + 1)
1/t^2
2/t^2
t^2
2*t^2
```

numer()

Returns the numerator of this element, as an element of the polynomial ring.

EXAMPLES:

```
sage: K = GF(11)['t'].fraction_field()
sage: t = K.gen(0); a = (t + 1/t)^3 - 1
sage: a.numer()
t^6 + 3*t^4 + 10*t^3 + 3*t^2 + 1
```

numerator()

Returns the numerator of this element, as an element of the polynomial ring.

EXAMPLES:

```
sage: K = GF(11)['t'].fraction_field()
sage: t = K.gen(0); a = (t + 1/t)^3 - 1
sage: a.numerator()
t^6 + 3*t^4 + 10*t^3 + 3*t^2 + 1
```

sqrt(*extend=True*, *all=False*)

Returns the square root of this element.

INPUT:

- `extend` - bool (default: True); if True, return a square root in an extension ring, if necessary. Otherwise, raise a ValueError if the square is not in the base ring.
- `all` - bool (default: False); if True, return all square roots of self, instead of just one.

EXAMPLES:

```
sage: from sage.rings.fraction_field_FpT import *
sage: K = GF(7)['t'].fraction_field(); t = K.gen(0)
sage: p = (t + 2)^2/(3*t^3 + 1)^4
sage: p.sqrt()
(3*t + 6)/(t^6 + 3*t^3 + 4)
sage: p.sqrt()^2 == p
True
```

subs (**args, **kwds*)

EXAMPLES:

```
sage: K = Frac(GF(11)['t'])
sage: t = K.gen()
sage: f = (t+1)/(t-1)
sage: f.subs(t=2)
3
sage: f.subs(X=2)
(t + 1)/(t + 10)
```

valuation (*v*)

Returns the valuation of self at v.

EXAMPLES:

```
sage: R.<t> = GF(5)[]
sage: f = (t+1)^2 * (t^2+t+1) / (t-1)^3
sage: f.valuation(t+1)
2
sage: f.valuation(t-1)
-3
sage: f.valuation(t)
0
```

class sage.rings.fraction_field_FpT.**FpT_Fp_section**

Bases: sage.categories.map.Section

This class represents the section from GF(p)(t) back to GF(p)[t]

EXAMPLES:

```
sage: R.<t> = GF(5)[]
sage: K = R.fraction_field()
sage: f = GF(5).convert_map_from(K); f
Section map:
  From: Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 5
  To:   Finite Field of size 5
sage: type(f)
<type 'sage.rings.fraction_field_FpT.FpT_Fp_section'>
```

class sage.rings.fraction_field_FpT.**FpT_Polyring_section**

Bases: sage.categories.map.Section

This class represents the section from GF(p)(t) back to GF(p)[t]

EXAMPLES:

```
sage: R.<t> = GF(5)[]
sage: K = R.fraction_field()
sage: f = R.convert_map_from(K); f
Section map:
  From: Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 5
```

```
  To:   Univariate Polynomial Ring in t over Finite Field of size 5
sage: type(f)
<type 'sage.rings.fraction_field_FpT.FpT_Polyring_section'>
```

class sage.rings.fraction_field_FpT.**FpT_iter**

Bases: object

Returns a class that iterates over all elements of an FpT.

EXAMPLES:

```
sage: K = GF(3)['t'].fraction_field()
sage: I = K.iter(1)
sage: list(I)
[0,
 1,
 2,
 t,
 t + 1,
 t + 2,
 2*t,
 2*t + 1,
 2*t + 2,
 1/t,
 2/t,
 (t + 1)/t,
 (t + 2)/t,
 (2*t + 1)/t,
 (2*t + 2)/t,
 1/(t + 1),
 2/(t + 1),
 t/(t + 1),
 (t + 2)/(t + 1),
 2*t/(t + 1),
 (2*t + 1)/(t + 1),
 1/(t + 2),
 2/(t + 2),
 t/(t + 2),
 (t + 1)/(t + 2),
 2*t/(t + 2),
 (2*t + 2)/(t + 2)]
```

next()

x.next() -> the next value, or raise StopIteration

class sage.rings.fraction_field_FpT.**Fp_FpT_coerce**

Bases: sage.rings.morphism.RingHomomorphism_coercion

This class represents the coercion map from GF(p) to GF(p)(t)

EXAMPLES:

```
sage: R.<t> = GF(5)[]
sage: K = R.fraction_field()
sage: f = K.coerce_map_from(GF(5)); f
Ring Coercion morphism:
  From: Finite Field of size 5
  To:   Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 5
sage: type(f)
<type 'sage.rings.fraction_field_FpT.Fp_FpT_coerce'>
```

section()

Returns the section of this inclusion: the partially defined map from `GF(p)(t)` back to `GF(p)`, defined on constant elements.

EXAMPLES:

```
sage: R.<t> = GF(5)[]
sage: K = R.fraction_field()
sage: f = K.coerce_map_from(GF(5))
sage: g = f.section(); g
Section map:
  From: Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 5
  To:   Finite Field of size 5
sage: t = K.gen()
sage: g(f(1,3,reduce=False))
2
sage: g(t)
Traceback (most recent call last):
...
ValueError: not constant
sage: g(1/t)
Traceback (most recent call last):
...
ValueError: not integral
```

class sage.rings.fraction_field_FpT.**Polyring_FpT_coerce**

Bases: `sage.rings.morphism.RingHomomorphism_coercion`

This class represents the coercion map from GF(p)[t] to GF(p)(t)

EXAMPLES:

```
sage: R.<t> = GF(5)[]
sage: K = R.fraction_field()
sage: f = K.coerce_map_from(R); f
Ring Coercion morphism:
  From: Univariate Polynomial Ring in t over Finite Field of size 5
  To:   Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 5
sage: type(f)
<type 'sage.rings.fraction_field_FpT.Polyring_FpT_coerce'>
```

section()

Returns the section of this inclusion: the partially defined map from `GF(p)(t)` back to `GF(p)[t]`, defined on elements with unit denominator.

EXAMPLES:

```
sage: R.<t> = GF(5)[]
sage: K = R.fraction_field()
sage: f = K.coerce_map_from(R)
sage: g = f.section(); g
Section map:
  From: Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 5
  To:   Univariate Polynomial Ring in t over Finite Field of size 5
sage: t = K.gen()
sage: g(t)
t
sage: g(1/t)
Traceback (most recent call last):
...
ValueError: not integral
```

class sage.rings.fraction_field_FpT.**ZZ_FpT_coerce**
Bases: sage.rings.morphism.RingHomomorphism_coercion

This class represents the coercion map from ZZ to GF(p)(t)

EXAMPLES:

```
sage: R.<t> = GF(17)[]
sage: K = R.fraction_field()
sage: f = K.coerce_map_from(ZZ); f
Ring Coercion morphism:
  From: Integer Ring
  To:   Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 17
sage: type(f)
<type 'sage.rings.fraction_field_FpT.ZZ_FpT_coerce'>
```

section()
Returns the section of this inclusion: the partially defined map from `GF(p)(t)` back to `ZZ`, defined on constant elements.

EXAMPLES:

```
sage: R.<t> = GF(5)[]
sage: K = R.fraction_field()
sage: f = K.coerce_map_from(ZZ)
sage: g = f.section(); g
Composite map:
  From: Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 5
  To:   Integer Ring
  Defn:   Section map:
          From: Fraction Field of Univariate Polynomial Ring in t over Finite Field of size
          To:   Finite Field of size 5
        then
          Lifting map:
          From: Finite Field of size 5
          To:   Integer Ring
sage: t = K.gen()
sage: g(f(1,3,reduce=False))
2
sage: g(t)
Traceback (most recent call last):
...
ValueError: not constant
sage: g(1/t)
Traceback (most recent call last):
...
ValueError: not integral
```

sage.rings.fraction_field_FpT.**unpickle_FpT_element**(*K*, *numer*, *denom*)
Used for pickling.

TESTS:

```
sage: from sage.rings.fraction_field_FpT import unpickle_FpT_element
sage: R.<t> = GF(13)['t']
sage: unpickle_FpT_element(Frac(R), t+1, t)
(t + 1)/t
```

CHAPTER

ELEVEN

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.

TESTS:

```
sage: R.<x> = PolynomialRing(ZZ)
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: S = R.quotient_ring(I);
```

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: class PowerIdeal(Ideal_nc):
....:     def __init__(self, R, n):
....:         self._power = n
....:         self._power = n
....:         Ideal_nc.__init__(self,R,[R.prod(m) for m in CartesianProduct(*[R.gens()]*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^3, y^2*z, y*z*x, y*z*y, y*z^2,
z*x^2, z*x*y, z*x*z, z*y*x, z*y^2, z*y*z, z^2*x, z^2*y, z^3) of
Free Algebra on 3 generators (x, y, z) over Rational Field
```

Free algebras have a custom quotient method that serves at creating finite dimensional quotients defined by multiplication matrices. We are bypassing it, so that we obtain the default quotient:

```
sage: 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^3, y^2*z, y*z*x, y*z*y, y*z^2, z*x^2, z*x*y,
z*x*z, z*y*x, z*y^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: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
sage: Q3 = F.quo(F*[F.prod(m) for m in CartesianProduct(*[F.gens()]*3)]*F)
sage: Q3
Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) over Rational Field by the idea
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 CartesianProduct(*[F.gens()]*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.reduce(x)` returning the normal form of elements $x \in R$. In other words, it is required that `I.reduce(x)==I.reduce(y)` $\iff x - y \in I$, and `x-I.reduce(x) in I`, for all $x, y \in R$.

EXAMPLES:

Some simple quotient rings with the integers:

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

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

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

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

sage: S = QuotientRing(QQ[x], QQ[x].ideal(x^2 - 2)); S
Univariate Quotient Polynomial Ring in xbar over Rational Field with
modulus x^2 - 2
sage: xbar = S.gen(); S.gen()
xbar
sage: for n in range(3): xbar^n
1
xbar
2
```

Sage coerces objects into ideals when possible:

```
sage: R = QuotientRing(QQ[x], x^2 + 1); R
Univariate Quotient Polynomial Ring in xbar over Rational Field with
modulus x^2 + 1
```

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

```
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
```

TESTS:

By trac ticket #11068, the following does not return a generic quotient ring but a usual quotient of the integer ring:

```
sage: R = Integers(8)
sage: I = R.ideal(2)
sage: R.quotient(I)
Ring of integers modulo 2
```

Here is an example of the quotient of a free algebra by a twosided homogeneous ideal (see trac ticket #7797):

```
sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
sage: I = F*[x*y+y*z,x^2+x*y-y*x-y^2]*F
sage: Q.<a,b,c> = F.quo(I); Q
Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) over Rational Field by the
sage: a*b
-b*c
sage: a^3
-b*c*a - b*c*b - b*c*c
sage: J = Q*[a^3-b^3]*Q
sage: R.<i,j,k> = Q.quo(J); R
Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) over Rational Field by the
sage: i^3
-j*k*i - j*k*j - j*k*k
sage: j^3
-j*k*i - j*k*j - j*k*k
```

Check that trac ticket #5978 is fixed by if we quotient by the zero ideal (0) then we just return `R`:

```
sage: R = QQ['x']
sage: R.quotient(R.zero_ideal())
Univariate Polynomial Ring in x over Rational Field
sage: R.<x> = PolynomialRing(ZZ)
sage: R is R.quotient(R.zero_ideal())
True
sage: I = R.ideal(0)
sage: R is R.quotient(I)
True
```

class `sage.rings.quotient_ring.`**`QuotientRing_generic`**(*R*, *I*, *names*, *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.

EXAMPLE:

```
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 +
```

class `sage.rings.quotient_ring.`**`QuotientRing_nc`**(*R*, *I*, *names*, *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 Rational Field by the
sage: a*b
-b*c
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 Rational Field by the
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:

```
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 +

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.

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

Element
alias of `QuotientRingElement`

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

EXAMPLES:

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

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

construction()

Returns the functorial construction of `self`.

EXAMPLES:

```
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) over Rational Fi
```

TESTS:

```
sage: F, R = Integers(5).construction()
sage: F(R)
Ring of integers modulo 5
sage: F, R = GF(5).construction()
sage: F(R)
Finite Field of size 5
```

cover()

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

EXAMPLES:

```
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
  To:   Multivariate Polynomial Ring in x, y over Rational Field
  Defn: Choice of lifting map
sage: l(x+y^3)
x
```

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: Q = QuotientRing(QQ[x], 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 ideal (x, y^2 +
sage: R.gen(0); R.gen(1)
x
y
sage: S.gen(0); S.gen(1)
a
b
sage: T.gen(0); T.gen(1)
0
d
```

ideal(**gens*, ***kwds*)

Return the ideal of `self` with the given generators.

EXAMPLES:

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

TESTS:

We create an ideal of a fairly generic integer ring (see trac ticket #5666):

```
sage: R = Integers(10)
sage: R.ideal(1)
Principal ideal (1) of Ring of integers modulo 10
```

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.

TESTS:

```
sage: Q = QuotientRing(ZZ,7*ZZ)
sage: Q.is_field()
True
```

Requires the `is_maximal` method of the defining ideal to be implemented:

```
sage: R.<x, y> = ZZ[]
sage: R.quotient_ring(R.ideal([2, 4 +x])).is_field()
Traceback (most recent call last):
...
NotImplementedError
```

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:

```
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:

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

sage: R = QuotientRing(QQ[x], 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:

```
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
  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
  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
  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 `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
(
  [0 1]
  [0 0],

  [0 0]
  [1 1]
)

  To:   Full MatrixSpace of 2 by 2 dense matrices over Finite Field of size 5
  Defn: Choice of lifting map
```

ngens()

Returns the number of generators for this quotient ring.

Todo

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

EXAMPLES:

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

sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = QuotientRing(R,R.ideal(1 + y^2))
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 +
sage: R.gens(); S.gens(); T.gens()
(x, y)
(a, b)
(0, d)
sage: R.ngens(); S.ngens(); T.ngens()
2
2
2
```

retract(*x*)

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

INPUT:

- x – An element of the cover ring

OUTPUT:

The image of the given element in self.

EXAMPLE:

```
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
```

CHAPTER

TWELVE

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
Quotient of Univariate Polynomial Ring in x over Integer Ring by the ideal (x^2 + 3*x + 4, x^2 +
sage: v = S.gens(); v
(xbar,)

sage: loads(v[0].dumps()) == v[0]
True

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

We name each of the generators.

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

is_unit()

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

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:

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

lc()

Return the leading coefficent of this quotient ring element.

EXAMPLE:

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

TESTS:

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

lift()

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

EXAMPLES:

```
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)
<class 'sage.rings.quotient_ring_element.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.

EXAMPLE:

```
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
```

TESTS:

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

lt()

Return the leading term of this quotient ring element.

EXAMPLE:

```
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
```

TESTS:

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

monomials()

Return the monomials in `self`.

OUTPUT:

A list of monomials.

EXAMPLES:

```
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)
<class 'sage.rings.quotient_ring_element.QuotientRing_generic_with_category.element_class'>
sage: a.monomials()
[a]
sage: (a+a*b).monomials()
[a*b, a]
sage: R.zero().monomials()
[]
```

reduce(*G*)

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

INPUT:

- `G` - a list of quotient ring elements

EXAMPLE:

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

variables()

Return all variables occurring in `self`.

OUTPUT:

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

EXAMPLES:

```
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)
<class 'sage.rings.quotient_ring_element.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)
```

CHAPTER

THIRTEEN

CLASSICAL INVARIANT THEORY

This module lists classical invariants and covariants of homogeneous polynomials (also called algebraic forms) under the action of the special linear group. That is, we are dealing with polynomials of degree d in n variables. The special linear group $SL(n, \mathbf{C})$ acts on the variables $(x_1, \ldots, x_n)$ linearly,

$$(x_1, \ldots, x_n)^t \to A(x_1, \ldots, x_n)^t, \qquad A \in SL(n, \mathbf{C})$$

The linear action on the variables transforms a polynomial p generally into a different polynomial gp. We can think of it as an action on the space of coefficients in p. An invariant is a polynomial in the coefficients that is invariant under this action. A covariant is a polynomial in the coefficients and the variables $(x_1, \ldots, x_n)$ that is invariant under the combined action.

For example, the binary quadratic $p(x, y) = ax^2 + bxy + cy^2$ has as its invariant the discriminant $disc(p) = b^2 - 4ac$. This means that for any $SL(2, \mathbf{C})$ coordinate change

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \qquad \alpha\delta - \beta\gamma = 1$$

the discriminant is invariant, $disc\big(p(x', y')\big) = disc\big(p(x, y)\big)$.

To use this module, you should use the factory object `invariant_theory`. For example, take the quartic:

```
sage: R.<x,y> = QQ[]
sage: q = x^4 + y^4
sage: quartic = invariant_theory.binary_quartic(q);  quartic
Binary quartic with coefficients (1, 0, 0, 0, 1)
```

One invariant of a quartic is known as the Eisenstein D-invariant. Since it is an invariant, it is a polynomial in the coefficients (which are integers in this example):

```
sage: quartic.EisensteinD()
1
```

One example of a covariant of a quartic is the so-called g-covariant (actually, the Hessian). As with all covariants, it is a polynomial in x, y and the coefficients:

```
sage: quartic.g_covariant()
-x^2*y^2
```

As usual, use tab completion and the online help to discover the implemented invariants and covariants.

In general, the variables of the defining polynomial cannot be guessed. For example, the zero polynomial can be thought of as a homogeneous polynomial of any degree. Also, since we also want to allow polynomial coefficients we cannot just take all variables of the polynomial ring as the variables of the form. This is why you will have to specify the variables explicitly if there is any potential ambiguity. For example:

```
sage: invariant_theory.binary_quartic(R.zero(), [x,y])
Binary quartic with coefficients (0, 0, 0, 0, 0)

sage: invariant_theory.binary_quartic(x^4, [x,y])
Binary quartic with coefficients (0, 0, 0, 0, 1)

sage: R.<x,y,t> = QQ[]
sage: invariant_theory.binary_quartic(x^4 + y^4 + t*x^2*y^2, [x,y])
Binary quartic with coefficients (1, 0, t, 0, 1)
```

Finally, it is often convenient to use inhomogeneous polynomials where it is understood that one wants to homogenize them. This is also supported, just define the form with an inhomogeneous polynomial and specify one less variable:

```
sage: R.<x,t> = QQ[]
sage: invariant_theory.binary_quartic(x^4 + 1 + t*x^2, [x])
Binary quartic with coefficients (1, 0, t, 0, 1)
```

REFERENCES:

class `sage.rings.invariant_theory.AlgebraicForm`(*n*, *d*, *polynomial*, **args*, ***kwds*)

Bases: `sage.rings.invariant_theory.FormsBase`

The base class of algebraic forms (i.e. homogeneous polynomials).

You should only instantiate the derived classes of this base class.

Derived classes must implement `coeffs()` and `scaled_coeffs()`

INPUT:

- `n` – The number of variables.
- `d` – The degree of the polynomial.
- `polynomial` – The polynomial.
- `*args` – The variables, as a single list/tuple, multiple arguments, or `None` to use all variables of the polynomial.

Derived classes must implement the same arguments for the constructor.

EXAMPLES:

```
sage: from sage.rings.invariant_theory import AlgebraicForm
sage: R.<x,y> = QQ[]
sage: p = x^2 + y^2
sage: AlgebraicForm(2, 2, p).variables()
(x, y)
sage: AlgebraicForm(2, 2, p, None).variables()
(x, y)
sage: AlgebraicForm(3, 2, p).variables()
(x, y, None)
sage: AlgebraicForm(3, 2, p, None).variables()
(x, y, None)

sage: from sage.rings.invariant_theory import AlgebraicForm
sage: R.<x,y,s,t> = QQ[]
sage: p = s*x^2 + t*y^2
sage: AlgebraicForm(2, 2, p, [x,y]).variables()
(x, y)
sage: AlgebraicForm(2, 2, p, x,y).variables()
(x, y)
```

```
sage: AlgebraicForm(3, 2, p, [x,y,None]).variables()
(x, y, None)
sage: AlgebraicForm(3, 2, p, x,y,None).variables()
(x, y, None)

sage: AlgebraicForm(2, 1, p, [x,y]).variables()
Traceback (most recent call last):
...
ValueError: Polynomial is of the wrong degree.

sage: AlgebraicForm(2, 2, x^2+y, [x,y]).variables()
Traceback (most recent call last):
...
ValueError: Polynomial is not homogeneous.
```

coefficients()

Alias for `coeffs()`.

See the documentation for `coeffs()` for details.

EXAMPLES:

```
sage: R.<a,b,c,d,e,f,g, x,y,z> = QQ[]
sage: p = a*x^2 + b*y^2 + c*z^2 + d*x*y + e*x*z + f*y*z
sage: q = invariant_theory.quadratic_form(p, x,y,z)
sage: q.coefficients()
(a, b, c, d, e, f)
sage: q.coeffs()
(a, b, c, d, e, f)
```

form()

Return the defining polynomial.

OUTPUT:

The polynomial used to define the algebraic form.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: quartic = invariant_theory.binary_quartic(x^4+y^4)
sage: quartic.form()
x^4 + y^4
sage: quartic.polynomial()
x^4 + y^4
```

homogenized(*var='h'*)

Return form as defined by a homogeneous polynomial.

INPUT:

- `var` – either a variable name, variable index or a variable (default: `'h'`).

OUTPUT:

The same algebraic form, but defined by a homogeneous polynomial.

EXAMPLES:

```
sage: T.<t> = QQ[]
sage: quadratic = invariant_theory.binary_quadratic(t^2 + 2*t + 3)
sage: quadratic
```

```
Binary quadratic with coefficients (1, 3, 2)
sage: quadratic.homogenized()
Binary quadratic with coefficients (1, 3, 2)
sage: quadratic == quadratic.homogenized()
True
sage: quadratic.form()
t^2 + 2*t + 3
sage: quadratic.homogenized().form()
t^2 + 2*t*h + 3*h^2

sage: R.<x,y,z> = QQ[]
sage: quadratic = invariant_theory.ternary_quadratic(x^2 + 1, [x,y])
sage: quadratic.homogenized().form()
x^2 + h^2
```

polynomial()

Return the defining polynomial.

OUTPUT:

The polynomial used to define the algebraic form.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: quartic = invariant_theory.binary_quartic(x^4+y^4)
sage: quartic.form()
x^4 + y^4
sage: quartic.polynomial()
x^4 + y^4
```

transformed(g)

Return the image under a linear transformation of the variables.

INPUT:

- **g – a $GL(n, \mathbf{C})$ matrix or a dictionary with the** variables as keys. A matrix is used to define the linear transformation of homogeneous variables, a dictionary acts by substitution of the variables.

OUTPUT:

A new instance of a subclass of `AlgebraicForm` obtained by replacing the variables of the homogeneous polynomial by their image under `g`.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: cubic = invariant_theory.ternary_cubic(x^3 + 2*y^3 + 3*z^3 + 4*x*y*z)
sage: cubic.transformed({x:y, y:z, z:x}).form()
3*x^3 + y^3 + 4*x*y*z + 2*z^3
sage: cyc = matrix([[0,1,0],[0,0,1],[1,0,0]])
sage: cubic.transformed(cyc) == cubic.transformed({x:y, y:z, z:x})
True
sage: g = matrix(QQ, [[1, 0, 0], [-1, 1, -3], [-5, -5, 16]])
sage: cubic.transformed(g)
Ternary cubic with coefficients (-356, -373, 12234, -1119, 3578, -1151,
3582, -11766, -11466, 7360)
sage: cubic.transformed(g).transformed(g.inverse()) == cubic
True
```

class sage.rings.invariant_theory.**BinaryQuartic**(*n*, *d*, *polynomial*, **args*)

Bases: sage.rings.invariant_theory.AlgebraicForm

Invariant theory of a binary quartic.

You should use the invariant_theory factory object to construct instances of this class. See binary_quartic() for details.

TESTS:

```
sage: R.<a0, a1, a2, a3, a4, x0, x1> = QQ[]
sage: p = a0*x1^4 + a1*x1^3*x0 + a2*x1^2*x0^2 + a3*x1*x0^3 + a4*x0^4
sage: quartic = invariant_theory.binary_quartic(p, x0, x1)
sage: quartic._check_covariant('form')
sage: quartic._check_covariant('EisensteinD', invariant=True)
sage: quartic._check_covariant('EisensteinE', invariant=True)
sage: quartic._check_covariant('g_covariant')
sage: quartic._check_covariant('h_covariant')
sage: TestSuite(quartic).run()
```

EisensteinD()

One of the Eisenstein invariants of a binary quartic.

OUTPUT:

The Eisenstein D-invariant of the quartic.

$$f(x) = a_0x_1^4 + 4a_1x_0x_1^3 + 6a_2x_0^2x_1^2 + 4a_3x_0^3x_1 + a_4x_0^4$$
$$\Rightarrow D(f) = a_0a_4 + 3a_2^2 - 4a_1a_3$$

EXAMPLES:

```
sage: R.<a0, a1, a2, a3, a4, x0, x1> = QQ[]
sage: f = a0*x1^4+4*a1*x0*x1^3+6*a2*x0^2*x1^2+4*a3*x0^3*x1+a4*x0^4
sage: inv = invariant_theory.binary_quartic(f, x0, x1)
sage: inv.EisensteinD()
3*a2^2 - 4*a1*a3 + a0*a4
```

EisensteinE()

One of the Eisenstein invariants of a binary quartic.

OUTPUT:

The Eisenstein E-invariant of the quartic.

$$f(x) = a_0x_1^4 + 4a_1x_0x_1^3 + 6a_2x_0^2x_1^2 + 4a_3x_0^3x_1 + a_4x_0^4$$
$$\Rightarrow E(f) = a_0a_3^2 + a_1^2a_4 - a_0a_2a_4 - 2a_1a_2a_3 + a_2^3$$

EXAMPLES:

```
sage: R.<a0, a1, a2, a3, a4, x0, x1> = QQ[]
sage: f = a0*x1^4+4*a1*x0*x1^3+6*a2*x0^2*x1^2+4*a3*x0^3*x1+a4*x0^4
sage: inv = invariant_theory.binary_quartic(f, x0, x1)
sage: inv.EisensteinE()
a2^3 - 2*a1*a2*a3 + a0*a3^2 + a1^2*a4 - a0*a2*a4
```

coeffs()

The coefficients of a binary quartic.

Given

$$f(x) = a_0x_1^4 + a_1x_0x_1^3 + a_2x_0^2x_1^2 + a_3x_0^3x_1 + a_4x_0^4$$

this function returns $a = (a_0, a_1, a_2, a_3, a_4)$

EXAMPLES:

```
sage: R.<a0, a1, a2, a3, a4, x0, x1> = QQ[]
sage: p = a0*x1^4 + a1*x1^3*x0 + a2*x1^2*x0^2 + a3*x1*x0^3 + a4*x0^4
sage: quartic = invariant_theory.binary_quartic(p, x0, x1)
sage: quartic.coeffs()
(a0, a1, a2, a3, a4)

sage: R.<a0, a1, a2, a3, a4, x> = QQ[]
sage: p = a0 + a1*x + a2*x^2 + a3*x^3 + a4*x^4
sage: quartic = invariant_theory.binary_quartic(p, x)
sage: quartic.coeffs()
(a0, a1, a2, a3, a4)
```

g_covariant()

The g-covariant of a binary quartic.

OUTPUT:

The g-covariant of the quartic.

$$f(x) = a_0 x_1^4 + 4a_1 x_0 x_1^3 + 6a_2 x_0^2 x_1^2 + 4a_3 x_0^3 x_1 + a_4 x_0^4$$
$$\Rightarrow D(f) = \frac{1}{144}\left(\frac{\partial^2 f}{\partial x \partial x}\right)$$

EXAMPLES:

```
sage: R.<a0, a1, a2, a3, a4, x, y> = QQ[]
sage: p = a0*x^4+4*a1*x^3*y+6*a2*x^2*y^2+4*a3*x*y^3+a4*y^4
sage: inv = invariant_theory.binary_quartic(p, x, y)
sage: g = inv.g_covariant();  g
a1^2*x^4 - a0*a2*x^4 + 2*a1*a2*x^3*y - 2*a0*a3*x^3*y + 3*a2^2*x^2*y^2
- 2*a1*a3*x^2*y^2 - a0*a4*x^2*y^2 + 2*a2*a3*x*y^3
- 2*a1*a4*x*y^3 + a3^2*y^4 - a2*a4*y^4

sage: inv_inhomogeneous = invariant_theory.binary_quartic(p.subs(y=1), x)
sage: inv_inhomogeneous.g_covariant()
a1^2*x^4 - a0*a2*x^4 + 2*a1*a2*x^3 - 2*a0*a3*x^3 + 3*a2^2*x^2
- 2*a1*a3*x^2 - a0*a4*x^2 + 2*a2*a3*x - 2*a1*a4*x + a3^2 - a2*a4

sage: g == 1/144 * (p.derivative(x,y)^2 - p.derivative(x,x)*p.derivative(y,y))
True
```

h_covariant()

The h-covariant of a binary quartic.

OUTPUT:

The h-covariant of the quartic.

$$f(x) = a_0 x_1^4 + 4a_1 x_0 x_1^3 + 6a_2 x_0^2 x_1^2 + 4a_3 x_0^3 x_1 + a_4 x_0^4$$
$$\Rightarrow D(f) = \frac{1}{144}\left(\frac{\partial^2 f}{\partial x \partial x}\right)$$

EXAMPLES:

```
sage: R.<a0, a1, a2, a3, a4, x, y> = QQ[]
sage: p = a0*x^4+4*a1*x^3*y+6*a2*x^2*y^2+4*a3*x*y^3+a4*y^4
sage: inv = invariant_theory.binary_quartic(p, x, y)
sage: h = inv.h_covariant();   h
```

```
-2*a1^3*x^6 + 3*a0*a1*a2*x^6 - a0^2*a3*x^6 - 6*a1^2*a2*x^5*y + 9*a0*a2^2*x^5*y
- 2*a0*a1*a3*x^5*y - a0^2*a4*x^5*y - 10*a1^2*a3*x^4*y^2 + 15*a0*a2*a3*x^4*y^2
- 5*a0*a1*a4*x^4*y^2 + 10*a0*a3^2*x^3*y^3 - 10*a1^2*a4*x^3*y^3
+ 10*a1*a3^2*x^2*y^4 - 15*a1*a2*a4*x^2*y^4 + 5*a0*a3*a4*x^2*y^4
+ 6*a2*a3^2*x*y^5 - 9*a2^2*a4*x*y^5 + 2*a1*a3*a4*x*y^5 + a0*a4^2*x*y^5
+ 2*a3^3*y^6 - 3*a2*a3*a4*y^6 + a1*a4^2*y^6

sage: inv_inhomogeneous = invariant_theory.binary_quartic(p.subs(y=1), x)
sage: inv_inhomogeneous.h_covariant()
-2*a1^3*x^6 + 3*a0*a1*a2*x^6 - a0^2*a3*x^6 - 6*a1^2*a2*x^5 + 9*a0*a2^2*x^5
- 2*a0*a1*a3*x^5 - a0^2*a4*x^5 - 10*a1^2*a3*x^4 + 15*a0*a2*a3*x^4
- 5*a0*a1*a4*x^4 + 10*a0*a3^2*x^3 - 10*a1^2*a4*x^3 + 10*a1*a3^2*x^2
- 15*a1*a2*a4*x^2 + 5*a0*a3*a4*x^2 + 6*a2*a3^2*x - 9*a2^2*a4*x
+ 2*a1*a3*a4*x + a0*a4^2*x + 2*a3^3 - 3*a2*a3*a4 + a1*a4^2

sage: g = inv.g_covariant()
sage: h == 1/8 * (p.derivative(x)*g.derivative(y)-p.derivative(y)*g.derivative(x))
True
```

monomials()

List the basis monomials in the form.

OUTPUT:

A tuple of monomials. They are in the same order as `coeffs()`.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: quartic = invariant_theory.binary_quartic(x^4+y^4)
sage: quartic.monomials()
(y^4, x*y^3, x^2*y^2, x^3*y, x^4)
```

scaled_coeffs()

The coefficients of a binary quartic.

Given

$$f(x) = a_0x_1^4 + 4a_1x_0x_1^3 + 6a_2x_0^2x_1^2 + 4a_3x_0^3x_1 + a_4x_0^4$$

this function returns $a = (a_0, a_1, a_2, a_3, a_4)$

EXAMPLES:

```
sage: R.<a0, a1, a2, a3, a4, x0, x1> = QQ[]
sage: quartic = a0*x1^4 + 4*a1*x1^3*x0 + 6*a2*x1^2*x0^2 + 4*a3*x1*x0^3 + a4*x0^4
sage: inv = invariant_theory.binary_quartic(quartic, x0, x1)
sage: inv.scaled_coeffs()
(a0, a1, a2, a3, a4)

sage: R.<a0, a1, a2, a3, a4, x> = QQ[]
sage: quartic = a0 + 4*a1*x + 6*a2*x^2 + 4*a3*x^3 + a4*x^4
sage: inv = invariant_theory.binary_quartic(quartic, x)
sage: inv.scaled_coeffs()
(a0, a1, a2, a3, a4)
```

class `sage.rings.invariant_theory.`**FormsBase**(*n*, *homogeneous*, *ring*, *variables*)

Bases: `sage.structure.sage_object.SageObject`

The common base class of `AlgebraicForm` and `SeveralAlgebraicForms`.

This is an abstract base class to provide common methods. It does not make much sense to instantiate it.

TESTS:

```
sage: from sage.rings.invariant_theory import FormsBase
sage: FormsBase(None, None, None, None)
<class 'sage.rings.invariant_theory.FormsBase'>
```

is_homogeneous()

Return whether the forms were defined by homogeneous polynomials.

OUTPUT:

Boolean. Whether the user originally defined the form via homogeneous variables.

EXAMPLES:

```
sage: R.<x,y,t> = QQ[]
sage: quartic = invariant_theory.binary_quartic(x^4+y^4+t*x^2*y^2, [x,y])
sage: quartic.is_homogeneous()
True
sage: quartic.form()
x^2*y^2*t + x^4 + y^4

sage: R.<x,y,t> = QQ[]
sage: quartic = invariant_theory.binary_quartic(x^4+1+t*x^2, [x])
sage: quartic.is_homogeneous()
False
sage: quartic.form()
x^4 + x^2*t + 1
```

ring()

Return the polynomial ring.

OUTPUT:

A polynomial ring. This is where the defining polynomial(s) live. Note that the polynomials may be homogeneous or inhomogeneous, depending on how the user constructed the object.

EXAMPLES:

```
sage: R.<x,y,t> = QQ[]
sage: quartic = invariant_theory.binary_quartic(x^4+y^4+t*x^2*y^2, [x,y])
sage: quartic.ring()
Multivariate Polynomial Ring in x, y, t over Rational Field

sage: R.<x,y,t> = QQ[]
sage: quartic = invariant_theory.binary_quartic(x^4+1+t*x^2, [x])
sage: quartic.ring()
Multivariate Polynomial Ring in x, y, t over Rational Field
```

variables()

Return the variables of the form.

OUTPUT:

A tuple of variables. If inhomogeneous notation is used for the defining polynomial then the last entry will be `None`.

EXAMPLES:

```
sage: R.<x,y,t> = QQ[]
sage: quartic = invariant_theory.binary_quartic(x^4+y^4+t*x^2*y^2, [x,y])
sage: quartic.variables()
```

```
(x, y)

sage: R.<x,y,t> = QQ[]
sage: quartic = invariant_theory.binary_quartic(x^4+1+t*x^2, [x])
sage: quartic.variables()
(x, None)
```

class sage.rings.invariant_theory.**InvariantTheoryFactory**

Bases: object

Factory object for invariants of multilinear forms.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: invariant_theory.ternary_cubic(x^3+y^3+z^3)
Ternary cubic with coefficients (1, 1, 1, 0, 0, 0, 0, 0, 0, 0)
```

binary_quadratic(*quadratic*, **args*)

Invariant theory of a quadratic in two variables.

INPUT:

- quadratic – a quadratic form.
- x, y – the homogeneous variables. If y is None, the quadratic is assumed to be inhomogeneous.

REFERENCES:

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: invariant_theory.binary_quadratic(x^2+y^2)
Binary quadratic with coefficients (1, 1, 0)

sage: T.<t> = QQ[]
sage: invariant_theory.binary_quadratic(t^2 + 2*t + 1, [t])
Binary quadratic with coefficients (1, 1, 2)
```

binary_quartic(*quartic*, **args*, ***kwds*)

Invariant theory of a quartic in two variables.

The algebra of invariants of a quartic form is generated by invariants i, j of degrees 2, 3. This ring is naturally isomorphic to the ring of modular forms of level 1, with the two generators corresponding to the Eisenstein series E_4 (see EisensteinD()) and E_6 (see EisensteinE()). The algebra of covariants is generated by these two invariants together with the form f of degree 1 and order 4, the Hessian g (see g_covariant()) of degree 2 and order 4, and a covariant h (see h_covariant()) of degree 3 and order 6. They are related by a syzygy

$$jf^3 - gf^2i + 4g^3 + h^2 = 0$$

of degree 6 and order 12.

INPUT:

- quartic – a quartic.
- x, y – the homogeneous variables. If y is None, the quartic is assumed to be inhomogeneous.

REFERENCES:

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: quartic = invariant_theory.binary_quartic(x^4+y^4)
sage: quartic
Binary quartic with coefficients (1, 0, 0, 0, 1)
sage: type(quartic)
<class 'sage.rings.invariant_theory.BinaryQuartic'>
```

inhomogeneous_quadratic_form(*polynomial*, **args*)

Invariants of an inhomogeneous quadratic form.

INPUT:

- `polynomial` – an inhomogeneous quadratic form.
- `*args` – the variables as multiple arguments, or as a single list/tuple.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: quadratic = x^2+2*y^2+3*x*y+4*x+5*y+6
sage: inv3 = invariant_theory.inhomogeneous_quadratic_form(quadratic)
sage: type(inv3)
<class 'sage.rings.invariant_theory.TernaryQuadratic'>
sage: inv4 = invariant_theory.inhomogeneous_quadratic_form(x^2+y^2+z^2)
sage: type(inv4)
<class 'sage.rings.invariant_theory.QuadraticForm'>
```

quadratic_form(*polynomial*, **args*)

Invariants of a homogeneous quadratic form.

INPUT:

- `polynomial` – a homogeneous or inhomogeneous quadratic form.
- `*args` – the variables as multiple arguments, or as a single list/tuple. If the last argument is `None`, the cubic is assumed to be inhomogeneous.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: quadratic = x^2+y^2+z^2
sage: inv = invariant_theory.quadratic_form(quadratic)
sage: type(inv)
<class 'sage.rings.invariant_theory.TernaryQuadratic'>
```

If some of the ring variables are to be treated as coefficients you need to specify the polynomial variables:

```
sage: R.<x,y,z, a,b> = QQ[]
sage: quadratic = a*x^2+b*y^2+z^2+2*y*z
sage: invariant_theory.quadratic_form(quadratic, x,y,z)
Ternary quadratic with coefficients (a, b, 1, 0, 0, 2)
sage: invariant_theory.quadratic_form(quadratic, [x,y,z])    # alternate syntax
Ternary quadratic with coefficients (a, b, 1, 0, 0, 2)
```

Inhomogeneous quadratic forms (see also `inhomogeneous_quadratic_form()`) can be specified by passing `None` as the last variable:

```
sage: inhom = quadratic.subs(z=1)
sage: invariant_theory.quadratic_form(inhom, x,y,None)
Ternary quadratic with coefficients (a, b, 1, 0, 0, 2)
```

quaternary_biquadratic(*quadratic1*, *quadratic2*, **args*, ***kwds*)

Invariants of two quadratics in four variables.

INPUT:

- `quadratic1`, `quadratic2` – two polynomias. Either homogeneous quadratic in 4 homogeneous variables, or inhomogeneous quadratic in 3 variables.
- `w`, `x`, `y`, `z` – the variables. If `z` is `None`, the quadratics are assumed to be inhomogeneous.

EXAMPLES:

```
sage: R.<w,x,y,z> = QQ[]
sage: q1 = w^2+x^2+y^2+z^2
sage: q2 = w*x + y*z
sage: inv = invariant_theory.quaternary_biquadratic(q1, q2)
sage: type(inv)
<class 'sage.rings.invariant_theory.TwoQuaternaryQuadratics'>
```

Distance between two spheres [Salmon]

```
sage: R.<x,y,z, a,b,c, r1,r2> = QQ[]
sage: S1 = -r1^2 + x^2 + y^2 + z^2
sage: S2 = -r2^2 + (x-a)^2 + (y-b)^2 + (z-c)^2
sage: inv = invariant_theory.quaternary_biquadratic(S1, S2, [x, y, z])
sage: inv.Delta_invariant()
-r1^2
sage: inv.Delta_prime_invariant()
-r2^2
sage: inv.Theta_invariant()
a^2 + b^2 + c^2 - 3*r1^2 - r2^2
sage: inv.Theta_prime_invariant()
a^2 + b^2 + c^2 - r1^2 - 3*r2^2
sage: inv.Phi_invariant()
2*a^2 + 2*b^2 + 2*c^2 - 3*r1^2 - 3*r2^2
sage: inv.J_covariant()
0
```

quaternary_quadratic(*quadratic*, **args*)

Invariant theory of a quadratic in four variables.

INPUT:

- `quadratic` – a quadratic form.
- `w`, `x`, `y`, `z` – the homogeneous variables. If `z` is `None`, the quadratic is assumed to be inhomogeneous.

REFERENCES:

EXAMPLES:

```
sage: R.<w,x,y,z> = QQ[]
sage: invariant_theory.quaternary_quadratic(w^2+x^2+y^2+z^2)
Quaternary quadratic with coefficients (1, 1, 1, 1, 0, 0, 0, 0, 0, 0)

sage: R.<x,y,z> = QQ[]
sage: invariant_theory.quaternary_quadratic(1+x^2+y^2+z^2)
Quaternary quadratic with coefficients (1, 1, 1, 1, 0, 0, 0, 0, 0, 0)
```

ternary_biquadratic(*quadratic1*, *quadratic2*, **args*, ***kwds*)

Invariants of two quadratics in three variables.

INPUT:

- `quadratic1`, `quadratic2` – two polynomials. Either homogeneous quadratic in 3 homogeneous variables, or inhomogeneous quadratic in 2 variables.
- `x`, `y`, `z` – the variables. If `z` is `None`, the quadratics are assumed to be inhomogeneous.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: q1 = x^2+y^2+z^2
sage: q2 = x*y + y*z + x*z
sage: inv = invariant_theory.ternary_biquadratic(q1, q2)
sage: type(inv)
<class 'sage.rings.invariant_theory.TwoTernaryQuadratics'>
```

Distance between two circles:

```
sage: R.<x,y, a,b, r1,r2> = QQ[]
sage: S1 = -r1^2 + x^2 + y^2
sage: S2 = -r2^2 + (x-a)^2 + (y-b)^2
sage: inv = invariant_theory.ternary_biquadratic(S1, S2, [x, y])
sage: inv.Delta_invariant()
-r1^2
sage: inv.Delta_prime_invariant()
-r2^2
sage: inv.Theta_invariant()
a^2 + b^2 - 2*r1^2 - r2^2
sage: inv.Theta_prime_invariant()
a^2 + b^2 - r1^2 - 2*r2^2
sage: inv.F_covariant()
2*x^2*a^2 + y^2*a^2 - 2*x*a^3 + a^4 + 2*x*y*a*b - 2*y*a^2*b + x^2*b^2 +
2*y^2*b^2 - 2*x*a*b^2 + 2*a^2*b^2 - 2*y*b^3 + b^4 - 2*x^2*r1^2 - 2*y^2*r1^2 +
2*x*a*r1^2 - 2*a^2*r1^2 + 2*y*b*r1^2 - 2*b^2*r1^2 + r1^4 - 2*x^2*r2^2 -
2*y^2*r2^2 + 2*x*a*r2^2 - 2*a^2*r2^2 + 2*y*b*r2^2 - 2*b^2*r2^2 + 2*r1^2*r2^2 +
r2^4
sage: inv.J_covariant()
-8*x^2*y*a^3 + 8*x*y*a^4 + 8*x^3*a^2*b - 16*x*y^2*a^2*b - 8*x^2*a^3*b +
8*y^2*a^3*b + 16*x^2*y*a*b^2 - 8*y^3*a*b^2 + 8*x*y^2*b^3 - 8*x^2*a*b^3 +
8*y^2*a*b^3 - 8*x*y*b^4 + 8*x*y*a^2*r1^2 - 8*y*a^3*r1^2 - 8*x^2*a*b*r1^2 +
8*y^2*a*b*r1^2 + 8*x*a^2*b*r1^2 - 8*x*y*b^2*r1^2 - 8*y*a*b^2*r1^2 + 8*x*b^3*r1^2 -
8*x*y*a^2*r2^2 + 8*x^2*a*b*r2^2 - 8*y^2*a*b*r2^2 + 8*x*y*b^2*r2^2
```

ternary_cubic(*cubic*, **args*, ***kwds*)

Invariants of a cubic in three variables.

The algebra of invariants of a ternary cubic under $SL_3(\mathbf{C})$ is a polynomial algebra generated by two invariants S (see `S_invariant()`) and T (see `T_invariant()`) of degrees 4 and 6, called Aronhold invariants.

The ring of covariants is given as follows. The identity covariant U of a ternary cubic has degree 1 and order 3. The Hessian H (see `Hessian()`) is a covariant of ternary cubics of degree 3 and order 3. There is a covariant Θ (see `Theta_covariant()`) of ternary cubics of degree 8 and order 6 that vanishes on points x lying on the Salmon conic of the polar of x with respect to the curve and its Hessian curve. The Brioschi covariant J (see `J_covariant()`) is the Jacobian of U, Θ, and H of degree 12, order 9. The algebra of covariants of a ternary cubic is generated over the ring of invariants by U, Θ, H, and J, with a relation

$$\begin{aligned} J^2 = & 4\Theta^3 + TU^2\Theta^2 + \Theta(-4S^3U^4 + 2STU^3H - 72S^2U^2H^2 \\ & - 18TUH^3 + 108SH^4) - 16S^4U^5H - 11S^2TU^4H^2 \\ & - 4T^2U^3H^3 + 54STU^2H^4 - 432S^2UH^5 - 27TH^6 \end{aligned}$$

REFERENCES:

INPUT:

- `cubic` – a homogeneous cubic in 3 homogeneous variables, or an inhomogeneous cubic in 2 variables.
- `x`, `y`, `z` – the variables. If `z` is `None`, the cubic is assumed to be inhomogeneous.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: cubic = invariant_theory.ternary_cubic(x^3+y^3+z^3)
sage: type(cubic)
<class 'sage.rings.invariant_theory.TernaryCubic'>
```

ternary_quadratic (*quadratic*, **args*, ***kwds*)

Invariants of a quadratic in three variables.

INPUT:

- `quadratic` – a homogeneous quadratic in 3 homogeneous variables, or an inhomogeneous quadratic in 2 variables.
- `x`, `y`, `z` – the variables. If `z` is `None`, the quadratic is assumed to be inhomogeneous.

REFERENCES:

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: invariant_theory.ternary_quadratic(x^2+y^2+z^2)
Ternary quadratic with coefficients (1, 1, 1, 0, 0, 0)

sage: T.<u, v> = QQ[]
sage: invariant_theory.ternary_quadratic(1+u^2+v^2)
Ternary quadratic with coefficients (1, 1, 1, 0, 0, 0)

sage: quadratic = x^2+y^2+z^2
sage: inv = invariant_theory.ternary_quadratic(quadratic)
sage: type(inv)
<class 'sage.rings.invariant_theory.TernaryQuadratic'>
```

class `sage.rings.invariant_theory.`**`QuadraticForm`** (*n*, *d*, *polynomial*, **args*)

Bases: `sage.rings.invariant_theory.AlgebraicForm`

Invariant theory of a multivariate quadratic form.

You should use the `invariant_theory` factory object to construct instances of this class. See `quadratic_form()` for details.

TESTS:

```
sage: R.<a,b,c,d,e,f,g, x,y,z> = QQ[]
sage: p = a*x^2 + b*y^2 + c*z^2 + d*x*y + e*x*z + f*y*z
sage: invariant_theory.quadratic_form(p, x,y,z)
Ternary quadratic with coefficients (a, b, c, d, e, f)
sage: type(_)
<class 'sage.rings.invariant_theory.TernaryQuadratic'>

sage: R.<a,b,c,d,e,f,g, x,y,z> = QQ[]
sage: p = a*x^2 + b*y^2 + c*z^2 + d*x*y + e*x*z + f*y*z
sage: invariant_theory.quadratic_form(p, x,y,z)
Ternary quadratic with coefficients (a, b, c, d, e, f)
```

```
sage: type(_)
<class 'sage.rings.invariant_theory.TernaryQuadratic'>
```

Since we cannot always decide whether the form is homogeneous or not based on the number of variables, you need to explicitly specify it if you want the variables to be treated as inhomogeneous:

```
sage: invariant_theory.inhomogeneous_quadratic_form(p.subs(z=1), x,y)
Ternary quadratic with coefficients (a, b, c, d, e, f)
```

as_QuadraticForm()

Convert into a `QuadraticForm`.

OUTPUT:

Sage has a special quadratic forms subsystem. This method converts `self` into this `QuadraticForm` representation.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: p = x^2+y^2+z^2+2*x*y+3*x*z
sage: quadratic = invariant_theory.ternary_quadratic(p)
sage: matrix(quadratic)
[  1   1 3/2]
[  1   1   0]
[3/2   0   1]
sage: quadratic.as_QuadraticForm()
Quadratic form in 3 variables over Multivariate Polynomial
Ring in x, y, z over Rational Field with coefficients:
[ 1/2 1 3/2 ]
[ * 1/2 0 ]
[ * * 1/2 ]
sage: _.polynomial('X,Y,Z')
X^2 + 2*X*Y + Y^2 + 3*X*Z + Z^2
```

coeffs()

The coefficients of a quadratic form.

Given

$$f(x) = \sum_{0\leq i<n} a_i x_i^2 + \sum_{0\leq j<k<n} a_{jk} x_j x_k$$

this function returns $a = (a_0, \dots, a_n, a_{00}, a_{01}, \dots, a_{n-1,n})$

EXAMPLES:

```
sage: R.<a,b,c,d,e,f,g, x,y,z> = QQ[]
sage: p = a*x^2 + b*y^2 + c*z^2 + d*x*y + e*x*z + f*y*z
sage: inv = invariant_theory.quadratic_form(p, x,y,z); inv
Ternary quadratic with coefficients (a, b, c, d, e, f)
sage: inv.coeffs()
(a, b, c, d, e, f)
sage: inv.scaled_coeffs()
(a, b, c, 1/2*d, 1/2*e, 1/2*f)
```

discriminant()

Return the discriminant of the quadratic form.

Up to an overall constant factor, this is just the determinant of the defining matrix, see `matrix()`. For a quadratic form in n variables, the overall constant is 2^{n-1} if n is odd and $(-1)^{n/2} 2^n$ if n is even.

EXAMPLES:

```
sage: R.<a,b,c, x,y> = QQ[]
sage: p = a*x^2+b*x*y+c*y^2
sage: quadratic = invariant_theory.quadratic_form(p, x,y)
sage: quadratic.discriminant()
b^2 - 4*a*c

sage: R.<a,b,c,d,e,f,g, x,y,z> = QQ[]
sage: p = a*x^2 + b*y^2 + c*z^2 + d*x*y + e*x*z + f*y*z
sage: quadratic = invariant_theory.quadratic_form(p, x,y,z)
sage: quadratic.discriminant()
4*a*b*c - c*d^2 - b*e^2 + d*e*f - a*f^2
```

dual()

Return the dual quadratic form.

OUTPUT:

A new quadratic form (with the same number of variables) defined by the adjoint matrix.

EXAMPLES:

```
sage: R.<a,b,c,x,y,z> = QQ[]
sage: cubic = x^2+y^2+z^2
sage: quadratic = invariant_theory.ternary_quadratic(a*x^2+b*y^2+c*z^2, [x,y,z])
sage: quadratic.form()
a*x^2 + b*y^2 + c*z^2
sage: quadratic.dual().form()
b*c*x^2 + a*c*y^2 + a*b*z^2

sage: R.<x,y,z, t> = QQ[]
sage: cubic = x^2+y^2+z^2
sage: quadratic = invariant_theory.ternary_quadratic(x^2+y^2+z^2 + t*x*y, [x,y,z])
sage: quadratic.dual()
Ternary quadratic with coefficients (1, 1, -1/4*t^2 + 1, -t, 0, 0)

sage: R.<x,y, t> = QQ[]
sage: quadratic = invariant_theory.ternary_quadratic(x^2+y^2+1 + t*x*y, [x,y])
sage: quadratic.dual()
Ternary quadratic with coefficients (1, 1, -1/4*t^2 + 1, -t, 0, 0)
```

TESTS:

```
sage: R = PolynomialRing(QQ, 'a20,a11,a02,a10,a01,a00,x,y,z', order='lex')
sage: R.inject_variables()
Defining a20, a11, a02, a10, a01, a00, x, y, z
sage: p = ( a20*x^2 + a11*x*y + a02*y^2 +
...         a10*x*z + a01*y*z + a00*z^2 )
sage: quadratic = invariant_theory.ternary_quadratic(p, x,y,z)
sage: quadratic.dual().dual().form().factor()
(1/4) *
(a20*x^2 + a11*x*y + a02*y^2 + a10*x*z + a01*y*z + a00*z^2) *
(4*a20*a02*a00 - a20*a01^2 - a11^2*a00 + a11*a10*a01 - a02*a10^2)

sage: R.<w,x,y,z> = QQ[]
sage: q = invariant_theory.quaternary_quadratic(w^2+2*x^2+3*y^2+4*z^2+x*y+5*w*z)
sage: q.form()
w^2 + 2*x^2 + x*y + 3*y^2 + 5*w*z + 4*z^2
sage: q.dual().dual().form().factor()
(42849/256) * (w^2 + 2*x^2 + x*y + 3*y^2 + 5*w*z + 4*z^2)
```

```
sage: R.<x,y,z> = QQ[]
sage: q = invariant_theory.quaternary_quadratic(1+2*x^2+3*y^2+4*z^2+x*y+5*z)
sage: q.form()
2*x^2 + x*y + 3*y^2 + 4*z^2 + 5*z + 1
sage: q.dual().dual().form().factor()
(42849/256) * (2*x^2 + x*y + 3*y^2 + 4*z^2 + 5*z + 1)
```

matrix()

Return the quadratic form as a symmetric matrix

OUTPUT:

This method returns a symmetric matrix A such that the quadratic Q equals

$$Q(x, y, z, \dots) = (x, y, \dots) A (x, y, \dots)^t$$

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: quadratic = invariant_theory.ternary_quadratic(x^2+y^2+z^2+x*y)
sage: matrix(quadratic)
[  1 1/2   0]
[1/2   1   0]
[  0   0   1]
sage: quadratic._matrix_() == matrix(quadratic)
True
```

monomials()

List the basis monomials in the form.

OUTPUT:

A tuple of monomials. They are in the same order as `coeffs()`.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: quadratic = invariant_theory.quadratic_form(x^2+y^2)
sage: quadratic.monomials()
(x^2, y^2, x*y)

sage: quadratic = invariant_theory.inhomogeneous_quadratic_form(x^2+y^2)
sage: quadratic.monomials()
(x^2, y^2, 1, x*y, x, y)
```

scaled_coeffs()

The scaled coefficients of a quadratic form.

Given

$$f(x) = \sum_{0 \leq i < n} a_i x_i^2 + \sum_{0 \leq j < k < n} 2 a_{jk} x_j x_k$$

this function returns $a = (a_0, \cdots, a_n, a_{00}, a_{01}, \dots, a_{n-1,n})$

EXAMPLES:

```
sage: R.<a,b,c,d,e,f,g, x,y,z> = QQ[]
sage: p = a*x^2 + b*y^2 + c*z^2 + d*x*y + e*x*z + f*y*z
sage: inv = invariant_theory.quadratic_form(p, x,y,z); inv
Ternary quadratic with coefficients (a, b, c, d, e, f)
```

```
sage: inv.coeffs()
(a, b, c, d, e, f)
sage: inv.scaled_coeffs()
(a, b, c, 1/2*d, 1/2*e, 1/2*f)
```

class sage.rings.invariant_theory.**SeveralAlgebraicForms**(*forms*)

Bases: sage.rings.invariant_theory.FormsBase

The base class of multiple algebraic forms (i.e. homogeneous polynomials).

You should only instantiate the derived classes of this base class.

See AlgebraicForm for the base class of a single algebraic form.

INPUT:

- forms – a list/tuple/iterable of at least one AlgebraicForm object, all with the same number of variables. Interpreted as multiple homogeneous polynomials in a common polynomial ring.

EXAMPLES:

```
sage: from sage.rings.invariant_theory import AlgebraicForm, SeveralAlgebraicForms
sage: R.<x,y> = QQ[]
sage: p = AlgebraicForm(2, 2, x^2, (x,y))
sage: q = AlgebraicForm(2, 2, y^2, (x,y))
sage: pq = SeveralAlgebraicForms([p, q])
```

get_form(*i*)

Return the i-th form.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: q1 = invariant_theory.quadratic_form(x^2 + y^2)
sage: q2 = invariant_theory.quadratic_form(x*y)
sage: from sage.rings.invariant_theory import SeveralAlgebraicForms
sage: q12 = SeveralAlgebraicForms([q1, q2])
sage: q12.get_form(0) is q1
True
sage: q12.get_form(1) is q2
True
sage: q12[0] is q12.get_form(0)   # syntactic sugar
True
sage: q12[1] is q12.get_form(1)   # syntactic sugar
True
```

homogenized(*var='h'*)

Return form as defined by a homogeneous polynomial.

INPUT:

- var – either a variable name, variable index or a variable (default: 'h').

OUTPUT:

The same algebraic form, but defined by a homogeneous polynomial.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: q = invariant_theory.quaternary_biquadratic(x^2+1, y^2+1, [x,y,z])
sage: q
Joint quaternary quadratic with coefficients (1, 0, 0, 1, 0, 0, 0, 0, 0, 0)
```

```
and quaternary quadratic with coefficients (0, 1, 0, 1, 0, 0, 0, 0, 0, 0)
sage: q.homogenized()
Joint quaternary quadratic with coefficients (1, 0, 0, 1, 0, 0, 0, 0, 0, 0)
and quaternary quadratic with coefficients (0, 1, 0, 1, 0, 0, 0, 0, 0, 0)
sage: type(q) is type(q.homogenized())
True
```

n_forms()

Return the number of forms.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: q1 = invariant_theory.quadratic_form(x^2 + y^2)
sage: q2 = invariant_theory.quadratic_form(x*y)
sage: from sage.rings.invariant_theory import SeveralAlgebraicForms
sage: q12 = SeveralAlgebraicForms([q1, q2])
sage: q12.n_forms()
2
sage: len(q12) == q12.n_forms()    # syntactic sugar
True
```

class sage.rings.invariant_theory.**TernaryCubic**(*n*, *d*, *polynomial*, **args*)

Bases: sage.rings.invariant_theory.AlgebraicForm

Invariant theory of a ternary cubic.

You should use the invariant_theory factory object to contstruct instances of this class. See ternary_cubic() for details.

TESTS:

```
sage: R.<x,y,z> = QQ[]
sage: cubic = invariant_theory.ternary_cubic(x^3+y^3+z^3)
sage: cubic
Ternary cubic with coefficients (1, 1, 1, 0, 0, 0, 0, 0, 0, 0)
sage: TestSuite(cubic).run()
```

Hessian()

Return the Hessian covariant.

OUTPUT:

The Hessian matrix multiplied with the conventional normalization factor $1/216$.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: cubic = invariant_theory.ternary_cubic(x^3+y^3+z^3)
sage: cubic.Hessian()
x*y*z

sage: R.<x,y> = QQ[]
sage: cubic = invariant_theory.ternary_cubic(x^3+y^3+1)
sage: cubic.Hessian()
x*y
```

J_covariant()

Return the J-covariant of the ternary cubic.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: cubic = invariant_theory.ternary_cubic(x^3+y^3+z^3)
sage: cubic.J_covariant()
x^6*y^3 - x^3*y^6 - x^6*z^3 + y^6*z^3 + x^3*z^6 - y^3*z^6

sage: R.<x,y> = QQ[]
sage: cubic = invariant_theory.ternary_cubic(x^3+y^3+1)
sage: cubic.J_covariant()
x^6*y^3 - x^3*y^6 - x^6 + y^6 + x^3 - y^3
```

S_invariant()

Return the S-invariant.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: cubic = invariant_theory.ternary_cubic(x^2*y+y^3+z^3+x*y*z)
sage: cubic.S_invariant()
-1/1296
```

T_invariant()

Return the T-invariant.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: cubic = invariant_theory.ternary_cubic(x^3+y^3+z^3)
sage: cubic.T_invariant()
1

sage: R.<x,y,z,t> = GF(7)[]
sage: cubic = invariant_theory.ternary_cubic(x^3+y^3+z^3+t*x*y*z, [x,y,z])
sage: cubic.T_invariant()
-t^6 - t^3 + 1
```

Theta_covariant()

Return the Θ covariant.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: cubic = invariant_theory.ternary_cubic(x^3+y^3+z^3)
sage: cubic.Theta_covariant()
-x^3*y^3 - x^3*z^3 - y^3*z^3

sage: R.<x,y> = QQ[]
sage: cubic = invariant_theory.ternary_cubic(x^3+y^3+1)
sage: cubic.Theta_covariant()
-x^3*y^3 - x^3 - y^3

sage: R.<x,y,z,a30,a21,a12,a03,a20,a11,a02,a10,a01,a00> = QQ[]
sage: p = ( a30*x^3 + a21*x^2*y + a12*x*y^2 + a03*y^3 + a20*x^2*z +
...         a11*x*y*z + a02*y^2*z + a10*x*z^2 + a01*y*z^2 + a00*z^3 )
sage: cubic = invariant_theory.ternary_cubic(p, x,y,z)
sage: len(list(cubic.Theta_covariant()))
6952
```

coeffs()

Return the coefficients of a cubic.

Given

$$p(x,y) = a_{30}x^3 + a_{21}x^2y + a_{12}xy^2 + a_{03}y^3 + a_{20}x^2 + \\ a_{11}xy + a_{02}y^2 + a_{10}x + a_{01}y + a_{00}$$

this function returns $a = (a_{30}, a_{03}, a_{00}, a_{21}, a_{20}, a_{12}, a_{02}, a_{10}, a_{01}, a_{11})$

EXAMPLES:

```
sage: R.<x,y,z,a30,a21,a12,a03,a20,a11,a02,a10,a01,a00> = QQ[]
sage: p = ( a30*x^3 + a21*x^2*y + a12*x*y^2 + a03*y^3 + a20*x^2*z +
...         a11*x*y*z + a02*y^2*z + a10*x*z^2 + a01*y*z^2 + a00*z^3 )
sage: invariant_theory.ternary_cubic(p, x,y,z).coeffs()
(a30, a03, a00, a21, a20, a12, a02, a10, a01, a11)
sage: invariant_theory.ternary_cubic(p.subs(z=1), x, y).coeffs()
(a30, a03, a00, a21, a20, a12, a02, a10, a01, a11)
```

monomials()

List the basis monomials of the form.

OUTPUT:

A tuple of monomials. They are in the same order as `coeffs()`.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: cubic = invariant_theory.ternary_cubic(x^3+y*z^2)
sage: cubic.monomials()
(x^3, y^3, z^3, x^2*y, x^2*z, x*y^2, y^2*z, x*z^2, y*z^2, x*y*z)
```

polar_conic()

Return the polar conic of the cubic.

OUTPUT:

Given the ternary cubic $f(X,Y,Z)$, this method returns the symmetric matrix $A(x,y,z)$ defined by

$$x f_X + y f_Y + z f_Z = (X,Y,Z) \cdot A(x,y,z) \cdot (X,Y,Z)^t$$

EXAMPLES:

```
sage: R.<x,y,z,X,Y,Z,a30,a21,a12,a03,a20,a11,a02,a10,a01,a00> = QQ[]
sage: p = ( a30*x^3 + a21*x^2*y + a12*x*y^2 + a03*y^3 + a20*x^2*z +
...         a11*x*y*z + a02*y^2*z + a10*x*z^2 + a01*y*z^2 + a00*z^3 )
sage: cubic = invariant_theory.ternary_cubic(p, x,y,z)
sage: cubic.polar_conic()
[  3*x*a30 + y*a21 + z*a20 x*a21 + y*a12 + 1/2*z*a11 x*a20 + 1/2*y*a11 + z*a10]
[x*a21 + y*a12 + 1/2*z*a11   x*a12 + 3*y*a03 + z*a02 1/2*x*a11 + y*a02 + z*a01]
[x*a20 + 1/2*y*a11 + z*a10 1/2*x*a11 + y*a02 + z*a01   x*a10 + y*a01 + 3*z*a00]

sage: polar_eqn = X*p.derivative(x) + Y*p.derivative(y) + Z*p.derivative(z)
sage: polar = invariant_theory.ternary_quadratic(polar_eqn, [x,y,z])
sage: polar.matrix().subs(X=x,Y=y,Z=z) == cubic.polar_conic()
True
```

scaled_coeffs()

Return the coefficients of a cubic.

Compared to `coeffs()`, this method returns rescaled coefficients that are often used in invariant theory.

Given

$$p(x, y) = a_{30}x^3 + a_{21}x^2y + a_{12}xy^2 + a_{03}y^3 + a_{20}x^2 + \\ a_{11}xy + a_{02}y^2 + a_{10}x + a_{01}y + a_{00}$$

this function returns $a = (a_{30}, a_{03}, a_{00}, a_{21}/3, a_{20}/3, a_{12}/3, a_{02}/3, a_{10}/3, a_{01}/3, a_{11}/6)$

EXAMPLES:

```
sage: R.<x,y,z,a30,a21,a12,a03,a20,a11,a02,a10,a01,a00> = QQ[]
sage: p = ( a30*x^3 + a21*x^2*y + a12*x*y^2 + a03*y^3 + a20*x^2*z +
...          a11*x*y*z + a02*y^2*z + a10*x*z^2 + a01*y*z^2 + a00*z^3 )
sage: invariant_theory.ternary_cubic(p, x,y,z).scaled_coeffs()
(a30, a03, a00, 1/3*a21, 1/3*a20, 1/3*a12, 1/3*a02, 1/3*a10, 1/3*a01, 1/6*a11)
```

syzygy (*U*, *S*, *T*, *H*, *Theta*, *J*)

Return the syzygy of the cubic evaluated on the invariants and covariants.

INPUT:

- `U`, `S`, `T`, `H`, `Theta`, `J` – polynomials from the same polynomial ring.

OUTPUT:

0 if evaluated for the form, the S invariant, the T invariant, the Hessian, the Θ covariant and the J-covariant of a ternary cubic.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: monomials = (x^3, y^3, z^3, x^2*y, x^2*z, x*y^2,
...                y^2*z, x*z^2, y*z^2, x*y*z)
sage: random_poly = sum([ randint(0,10000) * m for m in monomials ])
sage: cubic = invariant_theory.ternary_cubic(random_poly)
sage: U = cubic.form()
sage: S = cubic.S_invariant()
sage: T = cubic.T_invariant()
sage: H = cubic.Hessian()
sage: Theta = cubic.Theta_covariant()
sage: J = cubic.J_covariant()
sage: cubic.syzygy(U, S, T, H, Theta, J)
0
```

class `sage.rings.invariant_theory.`**`TernaryQuadratic`** (*n*, *d*, *polynomial*, **args*)

Bases: `sage.rings.invariant_theory.QuadraticForm`

Invariant theory of a ternary quadratic.

You should use the `invariant_theory` factory object to construct instances of this class. See `ternary_quadratic()` for details.

TESTS:

```
sage: R.<x,y,z> = QQ[]
sage: quadratic = invariant_theory.ternary_quadratic(x^2+y^2+z^2)
sage: quadratic
Ternary quadratic with coefficients (1, 1, 1, 0, 0, 0)
sage: TestSuite(quadratic).run()
```

coeffs ()

Return the coefficients of a quadratic.

Given

$$p(x, y) = a_{20}x^2 + a_{11}xy + a_{02}y^2 + a_{10}x + a_{01}y + a_{00}$$

this function returns $a = (a_{20}, a_{02}, a_{00}, a_{11}, a_{10}, a_{01})$

EXAMPLES:

```
sage: R.<x,y,z,a20,a11,a02,a10,a01,a00> = QQ[]
sage: p = ( a20*x^2 + a11*x*y + a02*y^2 +
...         a10*x*z + a01*y*z + a00*z^2 )
sage: invariant_theory.ternary_quadratic(p, x,y,z).coeffs()
(a20, a02, a00, a11, a10, a01)
sage: invariant_theory.ternary_quadratic(p.subs(z=1), x, y).coeffs()
(a20, a02, a00, a11, a10, a01)
```

covariant_conic(*other*)

Return the ternary quadratic covariant to `self` and `other`.

INPUT:

- `other` – Another ternary quadratic.

OUTPUT:

The so-called covariant conic, a ternary quadratic. It is symmetric under exchange of `self` and `other`.

EXAMPLES:

```
sage: ring.<x,y,z> = QQ[]
sage: Q = invariant_theory.ternary_quadratic(x^2+y^2+z^2)
sage: R = invariant_theory.ternary_quadratic(x*y+x*z+y*z)
sage: Q.covariant_conic(R)
-x*y - x*z - y*z
sage: R.covariant_conic(Q)
-x*y - x*z - y*z
```

TESTS:

```
sage: R.<a,a_,b,b_,c,c_,f,f_,g,g_,h,h_,x,y,z> = QQ[]
sage: p = ( a*x^2 + 2*h*x*y + b*y^2 +
...         2*g*x*z + 2*f*y*z + c*z^2 )
sage: Q = invariant_theory.ternary_quadratic(p, [x,y,z])
sage: Q.matrix()
[a h g]
[h b f]
[g f c]
sage: p = ( a_*x^2 + 2*h_*x*y + b_*y^2 +
...         2*g_*x*z + 2*f_*y*z + c_*z^2 )
sage: Q_ = invariant_theory.ternary_quadratic(p, [x,y,z])
sage: Q_.matrix()
[a_ h_ g_]
[h_ b_ f_]
[g_ f_ c_]
sage: QQ_ = Q.covariant_conic(Q_)
sage: invariant_theory.ternary_quadratic(QQ_, [x,y,z]).matrix()
[      b_*c + b*c_ - 2*f*f_  f_*g + f*g_ - c_*h - c*h_ -b_*g - b*g_ + f_*h + f*h_]
[ f_*g + f*g_ - c_*h - c*h_       a_*c + a*c_ - 2*g*g_ -a_*f - a*f_ + g_*h + g*h_]
[-b_*g - b*g_ + f_*h + f*h_ -a_*f - a*f_ + g_*h + g*h_       a_*b + a*b_ - 2*h*h_]
```

monomials()

List the basis monomials of the form.

OUTPUT:

A tuple of monomials. They are in the same order as `coeffs()`.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: quadratic = invariant_theory.ternary_quadratic(x^2+y*z)
sage: quadratic.monomials()
(x^2, y^2, z^2, x*y, x*z, y*z)
```

scaled_coeffs()

Return the scaled coefficients of a quadratic.

Given

$$p(x,y) = a_{20}x^2 + a_{11}xy + a_{02}y^2 + a_{10}x + a_{01}y + a_{00}$$

this function returns $a = (a_{20}, a_{02}, a_{00}, a_{11}/2, a_{10}/2, a_{01}/2,)$

EXAMPLES:

```
sage: R.<x,y,z,a20,a11,a02,a10,a01,a00> = QQ[]
sage: p = ( a20*x^2 + a11*x*y + a02*y^2 +
...         a10*x*z + a01*y*z + a00*z^2 )
sage: invariant_theory.ternary_quadratic(p, x,y,z).scaled_coeffs()
(a20, a02, a00, 1/2*a11, 1/2*a10, 1/2*a01)
sage: invariant_theory.ternary_quadratic(p.subs(z=1), x, y).scaled_coeffs()
(a20, a02, a00, 1/2*a11, 1/2*a10, 1/2*a01)
```

class sage.rings.invariant_theory.**TwoAlgebraicForms**(*forms*)

Bases: `sage.rings.invariant_theory.SeveralAlgebraicForms`

The Python constructor.

TESTS:

```
sage: from sage.rings.invariant_theory import AlgebraicForm, SeveralAlgebraicForms
sage: R.<x,y,z> = QQ[]
sage: p = AlgebraicForm(2, 2, x^2 + y^2)
sage: q = AlgebraicForm(2, 3, x^3 + y^3)
sage: r = AlgebraicForm(3, 3, x^3 + y^3 + z^3)
sage: pq = SeveralAlgebraicForms([p, q])
sage: pr = SeveralAlgebraicForms([p, r])
Traceback (most recent call last):
...
ValueError: All forms must be in the same variables.
```

first()

Return the first of the two forms.

OUTPUT:

The first algebraic form used in the definition.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: q0 = invariant_theory.quadratic_form(x^2 + y^2)
sage: q1 = invariant_theory.quadratic_form(x*y)
sage: from sage.rings.invariant_theory import TwoAlgebraicForms
sage: q = TwoAlgebraicForms([q0, q1])
sage: q.first() is q0
True
```

```
sage: q.get_form(0) is q0
True
sage: q.first().polynomial()
x^2 + y^2
```

second()

Return the second of the two forms.

OUTPUT:

The second form used in the definition.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: q0 = invariant_theory.quadratic_form(x^2 + y^2)
sage: q1 = invariant_theory.quadratic_form(x*y)
sage: from sage.rings.invariant_theory import TwoAlgebraicForms
sage: q = TwoAlgebraicForms([q0, q1])
sage: q.second() is q1
True
sage: q.get_form(1) is q1
True
sage: q.second().polynomial()
x*y
```

class sage.rings.invariant_theory.**TwoQuaternaryQuadratics**(*forms*)

Bases: sage.rings.invariant_theory.TwoAlgebraicForms

Invariant theory of two quaternary quadratics.

You should use the invariant_theory factory object to construct instances of this class. See quaternary_biquadratics() for details.

REFERENCES:

TESTS:

```
sage: R.<w,x,y,z> = QQ[]
sage: inv = invariant_theory.quaternary_biquadratic(w^2+x^2, y^2+z^2, w, x, y, z)
sage: inv
Joint quaternary quadratic with coefficients (1, 1, 0, 0, 0, 0, 0, 0, 0, 0) and
quaternary quadratic with coefficients (0, 0, 1, 1, 0, 0, 0, 0, 0, 0)
sage: TestSuite(inv).run()

sage: q1 = 73*x^2 + 96*x*y - 11*y^2 - 74*x*z - 10*y*z + 66*z^2 + 4*x + 63*y - 11*z + 57
sage: q2 = 61*x^2 - 100*x*y - 72*y^2 - 38*x*z + 85*y*z + 95*z^2 - 81*x + 39*y + 23*z - 7
sage: biquadratic = invariant_theory.quaternary_biquadratic(q1, q2, [x,y,z]).homogenized()
sage: biquadratic._check_covariant('Delta_invariant', invariant=True)
sage: biquadratic._check_covariant('Delta_prime_invariant', invariant=True)
sage: biquadratic._check_covariant('Theta_invariant', invariant=True)
sage: biquadratic._check_covariant('Theta_prime_invariant', invariant=True)
sage: biquadratic._check_covariant('Phi_invariant', invariant=True)
sage: biquadratic._check_covariant('T_covariant')
sage: biquadratic._check_covariant('T_prime_covariant')
sage: biquadratic._check_covariant('J_covariant')
```

Delta_invariant()

Return the Δ invariant.

EXAMPLES:

```
sage: R.<x,y,z,t,a0,a1,a2,a3,b0,b1,b2,b3,b4,b5,A0,A1,A2,A3,B0,B1,B2,B3,B4,B5> = QQ[]
sage: p1 = a0*x^2 + a1*y^2 + a2*z^2 + a3
sage: p1 += b0*x*y + b1*x*z + b2*x + b3*y*z + b4*y + b5*z
sage: p2 = A0*x^2 + A1*y^2 + A2*z^2 + A3
sage: p2 += B0*x*y + B1*x*z + B2*x + B3*y*z + B4*y + B5*z
sage: q = invariant_theory.quaternary_biquadratic(p1, p2, [x, y, z])
sage: coeffs = det(t * q[0].matrix() + q[1].matrix()).polynomial(t).coefficients(sparse=Fals
sage: q.Delta_invariant() == coeffs[4]
True
```

Delta_prime_invariant()

Return the Δ' invariant.

EXAMPLES:

```
sage: R.<x,y,z,t,a0,a1,a2,a3,b0,b1,b2,b3,b4,b5,A0,A1,A2,A3,B0,B1,B2,B3,B4,B5> = QQ[]
sage: p1 = a0*x^2 + a1*y^2 + a2*z^2 + a3
sage: p1 += b0*x*y + b1*x*z + b2*x + b3*y*z + b4*y + b5*z
sage: p2 = A0*x^2 + A1*y^2 + A2*z^2 + A3
sage: p2 += B0*x*y + B1*x*z + B2*x + B3*y*z + B4*y + B5*z
sage: q = invariant_theory.quaternary_biquadratic(p1, p2, [x, y, z])
sage: coeffs = det(t * q[0].matrix() + q[1].matrix()).polynomial(t).coefficients(sparse=Fals
sage: q.Delta_prime_invariant() == coeffs[0]
True
```

J_covariant()

The J-covariant.

This is the Jacobian determinant of the two biquadratics, the T-covariant, and the T'-covariant with respect to the four homogeneous variables.

EXAMPLES:

```
sage: R.<w,x,y,z,a0,a1,a2,a3,A0,A1,A2,A3> = QQ[]
sage: p1 = a0*x^2 + a1*y^2 + a2*z^2 + a3*w^2
sage: p2 = A0*x^2 + A1*y^2 + A2*z^2 + A3*w^2
sage: q = invariant_theory.quaternary_biquadratic(p1, p2, [w, x, y, z])
sage: q.J_covariant().factor()
z * y * x * w * (a3*A2 - a2*A3) * (a3*A1 - a1*A3) * (-a2*A1 + a1*A2)
* (a3*A0 - a0*A3) * (-a2*A0 + a0*A2) * (-a1*A0 + a0*A1)
```

Phi_invariant()

Return the Φ' invariant.

EXAMPLES:

```
sage: R.<x,y,z,t,a0,a1,a2,a3,b0,b1,b2,b3,b4,b5,A0,A1,A2,A3,B0,B1,B2,B3,B4,B5> = QQ[]
sage: p1 = a0*x^2 + a1*y^2 + a2*z^2 + a3
sage: p1 += b0*x*y + b1*x*z + b2*x + b3*y*z + b4*y + b5*z
sage: p2 = A0*x^2 + A1*y^2 + A2*z^2 + A3
sage: p2 += B0*x*y + B1*x*z + B2*x + B3*y*z + B4*y + B5*z
sage: q = invariant_theory.quaternary_biquadratic(p1, p2, [x, y, z])
sage: coeffs = det(t * q[0].matrix() + q[1].matrix()).polynomial(t).coefficients(sparse=Fals
sage: q.Phi_invariant() == coeffs[2]
True
```

T_covariant()

The T-covariant.

EXAMPLES:

```
sage: R.<x,y,z,t,a0,a1,a2,a3,b0,b1,b2,b3,b4,b5,A0,A1,A2,A3,B0,B1,B2,B3,B4,B5> = QQ[]
sage: p1 = a0*x^2 + a1*y^2 + a2*z^2 + a3
sage: p1 += b0*x*y + b1*x*z + b2*x + b3*y*z + b4*y + b5*z
sage: p2 = A0*x^2 + A1*y^2 + A2*z^2 + A3
sage: p2 += B0*x*y + B1*x*z + B2*x + B3*y*z + B4*y + B5*z
sage: q = invariant_theory.quaternary_biquadratic(p1, p2, [x, y, z])
sage: T = invariant_theory.quaternary_quadratic(q.T_covariant(), [x,y,z]).matrix()
sage: M = q[0].matrix().adjoint() + t*q[1].matrix().adjoint()
sage: M = M.adjoint().apply_map(             # long time (4s on my thinkpad W530)
....:         lambda m: m.coefficient(t))
sage: M == q.Delta_invariant()*T             # long time
True
```

T_prime_covariant()

The T'-covariant.

EXAMPLES:

```
sage: R.<x,y,z,t,a0,a1,a2,a3,b0,b1,b2,b3,b4,b5,A0,A1,A2,A3,B0,B1,B2,B3,B4,B5> = QQ[]
sage: p1 = a0*x^2 + a1*y^2 + a2*z^2 + a3
sage: p1 += b0*x*y + b1*x*z + b2*x + b3*y*z + b4*y + b5*z
sage: p2 = A0*x^2 + A1*y^2 + A2*z^2 + A3
sage: p2 += B0*x*y + B1*x*z + B2*x + B3*y*z + B4*y + B5*z
sage: q = invariant_theory.quaternary_biquadratic(p1, p2, [x, y, z])
sage: Tprime = invariant_theory.quaternary_quadratic(
....:     q.T_prime_covariant(), [x,y,z]).matrix()
sage: M = q[0].matrix().adjoint() + t*q[1].matrix().adjoint()
sage: M = M.adjoint().apply_map(                # long time (4s on my thinkpad W530)
....:         lambda m: m.coefficient(t^2))
sage: M == q.Delta_prime_invariant() * Tprime   # long time
True
```

Theta_invariant()

Return the Θ invariant.

EXAMPLES:

```
sage: R.<x,y,z,t,a0,a1,a2,a3,b0,b1,b2,b3,b4,b5,A0,A1,A2,A3,B0,B1,B2,B3,B4,B5> = QQ[]
sage: p1 = a0*x^2 + a1*y^2 + a2*z^2 + a3
sage: p1 += b0*x*y + b1*x*z + b2*x + b3*y*z + b4*y + b5*z
sage: p2 = A0*x^2 + A1*y^2 + A2*z^2 + A3
sage: p2 += B0*x*y + B1*x*z + B2*x + B3*y*z + B4*y + B5*z
sage: q = invariant_theory.quaternary_biquadratic(p1, p2, [x, y, z])
sage: coeffs = det(t * q[0].matrix() + q[1].matrix()).polynomial(t).coefficients(sparse=Fals
sage: q.Theta_invariant() == coeffs[3]
True
```

Theta_prime_invariant()

Return the Θ' invariant.

EXAMPLES:

```
sage: R.<x,y,z,t,a0,a1,a2,a3,b0,b1,b2,b3,b4,b5,A0,A1,A2,A3,B0,B1,B2,B3,B4,B5> = QQ[]
sage: p1 = a0*x^2 + a1*y^2 + a2*z^2 + a3
sage: p1 += b0*x*y + b1*x*z + b2*x + b3*y*z + b4*y + b5*z
sage: p2 = A0*x^2 + A1*y^2 + A2*z^2 + A3
sage: p2 += B0*x*y + B1*x*z + B2*x + B3*y*z + B4*y + B5*z
sage: q = invariant_theory.quaternary_biquadratic(p1, p2, [x, y, z])
sage: coeffs = det(t * q[0].matrix() + q[1].matrix()).polynomial(t).coefficients(sparse=Fals
sage: q.Theta_prime_invariant() == coeffs[1]
```

```
True
```

syzygy (*Delta*, *Theta*, *Phi*, *Theta_prime*, *Delta_prime*, *U*, *V*, *T*, *T_prime*, *J*)
Return the syzygy evaluated on the invariants and covariants.

INPUT:

- `Delta`, `Theta`, `Phi`, `Theta_prime`, `Delta_prime`, `U`, `V`, `T`, `T_prime`, `J` – polynomials from the same polynomial ring.

OUTPUT:

Zero if the `U` is the first polynomial, `V` the second polynomial, and the remaining input are the invariants and covariants of a quaternary biquadratic.

EXAMPLES:

```
sage: R.<w,x,y,z> = QQ[]
sage: monomials = [x^2, x*y, y^2, x*z, y*z, z^2, x*w, y*w, z*w, w^2]
sage: def q_rnd():  return sum(randint(-1000,1000)*m for m in monomials)
sage: biquadratic = invariant_theory.quaternary_biquadratic(q_rnd(), q_rnd())
sage: Delta = biquadratic.Delta_invariant()
sage: Theta = biquadratic.Theta_invariant()
sage: Phi = biquadratic.Phi_invariant()
sage: Theta_prime = biquadratic.Theta_prime_invariant()
sage: Delta_prime = biquadratic.Delta_prime_invariant()
sage: U = biquadratic.first().polynomial()
sage: V  = biquadratic.second().polynomial()
sage: T = biquadratic.T_covariant()
sage: T_prime  = biquadratic.T_prime_covariant()
sage: J = biquadratic.J_covariant()
sage: biquadratic.syzygy(Delta, Theta, Phi, Theta_prime, Delta_prime, U, V, T, T_prime, J)
0
```

If the arguments are not the invariants and covariants then the output is some (generically non-zero) polynomial:

```
sage: biquadratic.syzygy(1, 1, 1, 1, 1, 1, 1, 1, 1, x)
-x^2 + 1
```

class `sage.rings.invariant_theory.`**`TwoTernaryQuadratics`**(*forms*)
Bases: `sage.rings.invariant_theory.TwoAlgebraicForms`

Invariant theory of two ternary quadratics.

You should use the `invariant_theory` factory object to construct instances of this class. See `ternary_biquadratics()` for details.

REFERENCES:

TESTS:

```
sage: R.<x,y,z> = QQ[]
sage: inv = invariant_theory.ternary_biquadratic(x^2+y^2+z^2, x*y+y*z+x*z, [x, y, z])
sage: inv
Joint ternary quadratic with coefficients (1, 1, 1, 0, 0, 0) and ternary
quadratic with coefficients (0, 0, 0, 1, 1, 1)
sage: TestSuite(inv).run()

sage: q1 = 73*x^2 + 96*x*y - 11*y^2 + 4*x + 63*y + 57
sage: q2 = 61*x^2 - 100*x*y - 72*y^2 - 81*x + 39*y - 7
sage: biquadratic = invariant_theory.ternary_biquadratic(q1, q2, [x,y]).homogenized()
```

```
sage: biquadratic._check_covariant('Delta_invariant', invariant=True)
sage: biquadratic._check_covariant('Delta_prime_invariant', invariant=True)
sage: biquadratic._check_covariant('Theta_invariant', invariant=True)
sage: biquadratic._check_covariant('Theta_prime_invariant', invariant=True)
sage: biquadratic._check_covariant('F_covariant')
sage: biquadratic._check_covariant('J_covariant')
```

Delta_invariant()

Return the Δ invariant.

EXAMPLES:

```
sage: R.<a00, a01, a11, a02, a12, a22, b00, b01, b11, b02, b12, b22, y0, y1, y2, t> = QQ[]
sage: p1 = a00*y0^2 + 2*a01*y0*y1 + a11*y1^2 + 2*a02*y0*y2 + 2*a12*y1*y2 + a22*y2^2
sage: p2 = b00*y0^2 + 2*b01*y0*y1 + b11*y1^2 + 2*b02*y0*y2 + 2*b12*y1*y2 + b22*y2^2
sage: q = invariant_theory.ternary_biquadratic(p1, p2, [y0, y1, y2])
sage: coeffs = det(t * q[0].matrix() + q[1].matrix()).polynomial(t).coefficients(sparse=Fals
sage: q.Delta_invariant() == coeffs[3]
True
```

Delta_prime_invariant()

Return the Δ' invariant.

EXAMPLES:

```
sage: R.<a00, a01, a11, a02, a12, a22, b00, b01, b11, b02, b12, b22, y0, y1, y2, t> = QQ[]
sage: p1 = a00*y0^2 + 2*a01*y0*y1 + a11*y1^2 + 2*a02*y0*y2 + 2*a12*y1*y2 + a22*y2^2
sage: p2 = b00*y0^2 + 2*b01*y0*y1 + b11*y1^2 + 2*b02*y0*y2 + 2*b12*y1*y2 + b22*y2^2
sage: q = invariant_theory.ternary_biquadratic(p1, p2, [y0, y1, y2])
sage: coeffs = det(t * q[0].matrix() + q[1].matrix()).polynomial(t).coefficients(sparse=Fals
sage: q.Delta_prime_invariant() == coeffs[0]
True
```

F_covariant()

Return the F covariant.

EXAMPLES:

```
sage: R.<a00, a01, a11, a02, a12, a22, b00, b01, b11, b02, b12, b22, x, y> = QQ[]
sage: p1 = 73*x^2 + 96*x*y - 11*y^2 + 4*x + 63*y + 57
sage: p2 = 61*x^2 - 100*x*y - 72*y^2 - 81*x + 39*y - 7
sage: q = invariant_theory.ternary_biquadratic(p1, p2, [x, y])
sage: q.F_covariant()
-32566577*x^2 + 29060637/2*x*y + 20153633/4*y^2 -
30250497/2*x - 241241273/4*y - 323820473/16
```

J_covariant()

Return the J covariant.

EXAMPLES:

```
sage: R.<a00, a01, a11, a02, a12, a22, b00, b01, b11, b02, b12, b22, x, y> = QQ[]
sage: p1 = 73*x^2 + 96*x*y - 11*y^2 + 4*x + 63*y + 57
sage: p2 = 61*x^2 - 100*x*y - 72*y^2 - 81*x + 39*y - 7
sage: q = invariant_theory.ternary_biquadratic(p1, p2, [x, y])
sage: q.J_covariant()
1057324024445*x^3 + 1209531088209*x^2*y + 942116599708*x*y^2 +
984553030871*y^3 + 543715345505/2*x^2 - 3065093506021/2*x*y +
755263948570*y^2 - 1118430692650*x - 509948695327/4*y + 3369951531745/8
```

Theta_invariant()

Return the Θ invariant.

EXAMPLES:

```
sage: R.<a00, a01, a11, a02, a12, a22, b00, b01, b11, b02, b12, b22, y0, y1, y2, t> = QQ[]
sage: p1 = a00*y0^2 + 2*a01*y0*y1 + a11*y1^2 + 2*a02*y0*y2 + 2*a12*y1*y2 + a22*y2^2
sage: p2 = b00*y0^2 + 2*b01*y0*y1 + b11*y1^2 + 2*b02*y0*y2 + 2*b12*y1*y2 + b22*y2^2
sage: q = invariant_theory.ternary_biquadratic(p1, p2, [y0, y1, y2])
sage: coeffs = det(t * q[0].matrix() + q[1].matrix()).polynomial(t).coefficients(sparse=Fals
sage: q.Theta_invariant() == coeffs[2]
True
```

Theta_prime_invariant()

Return the Θ' invariant.

EXAMPLES:

```
sage: R.<a00, a01, a11, a02, a12, a22, b00, b01, b11, b02, b12, b22, y0, y1, y2, t> = QQ[]
sage: p1 = a00*y0^2 + 2*a01*y0*y1 + a11*y1^2 + 2*a02*y0*y2 + 2*a12*y1*y2 + a22*y2^2
sage: p2 = b00*y0^2 + 2*b01*y0*y1 + b11*y1^2 + 2*b02*y0*y2 + 2*b12*y1*y2 + b22*y2^2
sage: q = invariant_theory.ternary_biquadratic(p1, p2, [y0, y1, y2])
sage: coeffs = det(t * q[0].matrix() + q[1].matrix()).polynomial(t).coefficients(sparse=Fals
sage: q.Theta_prime_invariant() == coeffs[1]
True
```

syzygy(*Delta*, *Theta*, *Theta_prime*, *Delta_prime*, *S*, *S_prime*, *F*, *J*)

Return the syzygy evaluated on the invariants and covariants.

INPUT:

- Delta, Theta, Theta_prime, Delta_prime, S, S_prime, F, J – polynomials from the same polynomial ring.

OUTPUT:

Zero if S is the first polynomial, S_prime the second polynomial, and the remaining input are the invariants and covariants of a ternary biquadratic.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: monomials = [x^2, x*y, y^2, x*z, y*z, z^2]
sage: def q_rnd():  return sum(randint(-1000,1000)*m for m in monomials)
sage: biquadratic = invariant_theory.ternary_biquadratic(q_rnd(), q_rnd(), [x,y,z])
sage: Delta = biquadratic.Delta_invariant()
sage: Theta = biquadratic.Theta_invariant()
sage: Theta_prime = biquadratic.Theta_prime_invariant()
sage: Delta_prime = biquadratic.Delta_prime_invariant()
sage: S = biquadratic.first().polynomial()
sage: S_prime  = biquadratic.second().polynomial()
sage: F = biquadratic.F_covariant()
sage: J = biquadratic.J_covariant()
sage: biquadratic.syzygy(Delta, Theta, Theta_prime, Delta_prime, S, S_prime, F, J)
0
```

If the arguments are not the invariants and covariants then the output is some (generically non-zero) polynomial:

```
sage: biquadratic.syzygy(1, 1, 1, 1, 1, 1, 1, x)
1/64*x^2 + 1
```

CHAPTER FOURTEEN

CYTHON WRAPPER FOR BERNMM LIBRARY

AUTHOR:

- David Harvey (2008-06): initial version

`sage.rings.bernmm.`**`bernmm_bern_modp`**(*p*, *k*)

Computes $B_k \mod p$, where B_k is the k-th Bernoulli number.

If B_k is not p-integral, returns -1.

INPUT:

p – a prime k – non-negative integer

COMPLEXITY:

Pretty much linear in \$p\$.

EXAMPLES:

```
sage: from sage.rings.bernmm import bernmm_bern_modp

sage: bernoulli(0) % 5, bernmm_bern_modp(5, 0)
(1, 1)
sage: bernoulli(1) % 5, bernmm_bern_modp(5, 1)
(2, 2)
sage: bernoulli(2) % 5, bernmm_bern_modp(5, 2)
(1, 1)
sage: bernoulli(3) % 5, bernmm_bern_modp(5, 3)
(0, 0)
sage: bernoulli(4), bernmm_bern_modp(5, 4)
(-1/30, -1)
sage: bernoulli(18) % 5, bernmm_bern_modp(5, 18)
(4, 4)
sage: bernoulli(19) % 5, bernmm_bern_modp(5, 19)
(0, 0)

sage: p = 10000019; k = 1000
sage: bernoulli(k) % p
1972762
sage: bernmm_bern_modp(p, k)
1972762
```

`sage.rings.bernmm.`**`bernmm_bern_rat`**(*k*, *num_threads=1*)

Computes k-th Bernoulli number using a multimodular algorithm. (Wrapper for bernmm library.)

INPUT:

- k – non-negative integer

- num_threads – integer >= 1, number of threads to use

COMPLEXITY:

Pretty much quadratic in k. See the paper "A multimodular algorithm for computing Bernoulli numbers", David Harvey, 2008, for more details.

EXAMPLES:

```
sage: from sage.rings.bernmm import bernmm_bern_rat

sage: bernmm_bern_rat(0)
1
sage: bernmm_bern_rat(1)
-1/2
sage: bernmm_bern_rat(2)
1/6
sage: bernmm_bern_rat(3)
0
sage: bernmm_bern_rat(100)
-94598037819122125295227433069493721872702841533066936133385696204311395415197247711/33330
sage: bernmm_bern_rat(100, 3)
-94598037819122125295227433069493721872702841533066936133385696204311395415197247711/33330
```

TESTS:

```
sage: lst1 = [ bernoulli(2*k, algorithm='bernmm', num_threads=2) for k in [2932, 2957, 3443, 396
sage: lst2 = [ bernoulli(2*k, algorithm='pari') for k in [2932, 2957, 3443, 3962, 3973] ]
sage: lst1 == lst2
True
sage: [ Zmod(101)(t) for t in lst1 ]
[77, 72, 89, 98, 86]
sage: [ Zmod(101)(t) for t in lst2 ]
[77, 72, 89, 98, 86]
```

CHAPTER

FIFTEEN

BERNOULLI NUMBERS MODULO P

AUTHOR:

- David Harvey (2006-07-26): initial version
- William Stein (2006-07-28): some touch up.
- David Harvey (2006-08-06): new, faster algorithm, also using faster NTL interface
- David Harvey (2007-08-31): algorithm for a single Bernoulli number mod p
- David Harvey (2008-06): added interface to bernmm, removed old code

`sage.rings.bernoulli_mod_p.`**`bernoulli_mod_p`**`(p)`

Returns the bernoulli numbers $B_0, B_2, ...B_{p-3}$ modulo p.

INPUT:

p – integer, a prime

OUTPUT:

list – Bernoulli numbers modulo p as a list of integers [B(0), B(2), ... B(p-3)].

ALGORITHM:

Described in accompanying latex file.

PERFORMANCE:

Should be complexity $O(p \log p)$.

EXAMPLES:

Check the results against PARI's C-library implementation (that computes exact rationals) for $p = 37$:

```
sage: bernoulli_mod_p(37)
[1, 31, 16, 15, 16, 4, 17, 32, 22, 31, 15, 15, 17, 12, 29, 2, 0, 2]
sage: [bernoulli(n) % 37 for n in xrange(0, 36, 2)]
[1, 31, 16, 15, 16, 4, 17, 32, 22, 31, 15, 15, 17, 12, 29, 2, 0, 2]
```

Boundary case:

```
sage: bernoulli_mod_p(3)
[1]
```

AUTHOR:

– David Harvey (2006-08-06)

`sage.rings.bernoulli_mod_p.`**`bernoulli_mod_p_single`**`(p, k)`

Returns the bernoulli number B_k mod p.

If B_k is not p-integral, an ArithmeticError is raised.

INPUT:

p – integer, a prime k – non-negative integer

OUTPUT:

The k-th bernoulli number mod p.

EXAMPLES:

```
sage: bernoulli_mod_p_single(1009, 48)
628
sage: bernoulli(48) % 1009
628

sage: bernoulli_mod_p_single(1, 5)
Traceback (most recent call last):
...
ValueError: p (=1) must be a prime >= 3

sage: bernoulli_mod_p_single(100, 4)
Traceback (most recent call last):
...
ValueError: p (=100) must be a prime

sage: bernoulli_mod_p_single(19, 5)
0

sage: bernoulli_mod_p_single(19, 18)
Traceback (most recent call last):
...
ArithmeticError: B_k is not integral at p

sage: bernoulli_mod_p_single(19, -4)
Traceback (most recent call last):
...
ValueError: k must be non-negative
```

Check results against bernoulli_mod_p:

```
sage: bernoulli_mod_p(37)
 [1, 31, 16, 15, 16, 4, 17, 32, 22, 31, 15, 15, 17, 12, 29, 2, 0, 2]
sage: [bernoulli_mod_p_single(37, n) % 37 for n in xrange(0, 36, 2)]
 [1, 31, 16, 15, 16, 4, 17, 32, 22, 31, 15, 15, 17, 12, 29, 2, 0, 2]

sage: bernoulli_mod_p(31)
 [1, 26, 1, 17, 1, 9, 11, 27, 14, 23, 13, 22, 14, 8, 14]
sage: [bernoulli_mod_p_single(31, n) % 31 for n in xrange(0, 30, 2)]
 [1, 26, 1, 17, 1, 9, 11, 27, 14, 23, 13, 22, 14, 8, 14]

sage: bernoulli_mod_p(3)
 [1]
sage: [bernoulli_mod_p_single(3, n) % 3 for n in xrange(0, 2, 2)]
 [1]

sage: bernoulli_mod_p(5)
 [1, 1]
sage: [bernoulli_mod_p_single(5, n) % 5 for n in xrange(0, 4, 2)]
 [1, 1]
```

```
sage: bernoulli_mod_p(7)
[1, 6, 3]
sage: [bernoulli_mod_p_single(7, n) % 7 for n in xrange(0, 6, 2)]
[1, 6, 3]
```

AUTHOR:

– David Harvey (2007-08-31) – David Harvey (2008-06): rewrote to use bernmm library

`sage.rings.bernoulli_mod_p.`**`verify_bernoulli_mod_p`**(*data*)

Computes checksum for bernoulli numbers.

It checks the identity

$$\sum_{n=0}^{(p-3)/2} 2^{2n}(2n+1)B_{2n} \equiv -2 \pmod{p}$$

(see "Irregular Primes to One Million", Buhler et al)

INPUT:

data – list, same format as output of bernoulli_mod_p function

OUTPUT:

bool – True if checksum passed

EXAMPLES:

```
sage: from sage.rings.bernoulli_mod_p import verify_bernoulli_mod_p
sage: verify_bernoulli_mod_p(bernoulli_mod_p(next_prime(3)))
True
sage: verify_bernoulli_mod_p(bernoulli_mod_p(next_prime(1000)))
True
sage: verify_bernoulli_mod_p([1, 2, 4, 5, 4])
True
sage: verify_bernoulli_mod_p([1, 2, 3, 4, 5])
False
```

This one should test that long longs are working:

```
sage: verify_bernoulli_mod_p(bernoulli_mod_p(next_prime(20000)))
True
```

AUTHOR: David Harvey

CHAPTER

SIXTEEN

BIG O FOR VARIOUS TYPES (POWER SERIES, P-ADICS, ETC.)

sage.rings.big_oh.**O**(*x*)

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 in.

```
sage: R.<x> = QQ['x']
sage: O(x^100)
O(x^100)
sage: 1/(1+x+O(x^5))
1 - x + x^2 - x^3 + x^4 + O(x^5)
sage: R.<u,v> = QQ[[]]
sage: 1 + u + v^2 + O(u, v)^5
1 + u + v^2 + O(u, v)^5
```

This is also useful to create p-adic numbers.

```
sage: O(7^6)
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 a big_oh.

```
sage: 11^-12 + O(11^15)
11^-12 + O(11^8)
```

The reason that this fails is that the O function 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.

```
sage: K = Qp(11, 30)
sage: K(11^-12) + O(11^15)
11^-12 + O(11^15)
sage: 11^-12 + K(O(11^15))
```

```
11^-12 + O(11^15)
sage: K(11^-12, absprec = 15)
11^-12 + O(11^15)
sage: K(11^-12, 15)
11^-12 + O(11^15)
```

CHAPTER

SEVENTEEN

CONTINUED FRACTION FIELD

Sage implements the field `ContinuedFractionField` (or `CFF` for short) of finite simple continued fractions. This is really isomorphic to the field $\mathbf{Q}$ of rational numbers, but with different printing and semantics. It should be possible to use this field in most cases where one could use $\mathbf{Q}$, except arithmetic is *much* slower.

EXAMPLES:

We can create matrices, polynomials, vectors, etc., over the continued fraction field:

```
sage: a = random_matrix(CFF, 4)
sage: a
[    [-1; 2] [-1; 1, 94]       [0; 2]        [-12]]
[       [-1]       [0; 2]  [-1; 1, 3]   [0; 1, 2]]
[    [-3; 2]          [0]   [0; 1, 2]         [-1]]
[        [1]         [-1]      [0; 3]          [1]]
sage: f = a.charpoly()
sage: f
[1]*x^4 + ([-2; 3])*x^3 + [14; 1, 1, 1, 9, 1, 8]*x^2 + ([-13; 4, 1, 2, 1, 1, 1, 1, 1, 2, 2])*x + [-6;
sage: f(a)
[[0] [0] [0] [0]]
[[0] [0] [0] [0]]
[[0] [0] [0] [0]]
[[0] [0] [0] [0]]
sage: vector(CFF, [1/2, 2/3, 3/4, 4/5])
([0; 2], [0; 1, 2], [0; 1, 3], [0; 1, 4])
```

AUTHORS:

- Niles Johnson (2010-08): `random_element()` should pass on `*args` and `**kwds` (trac ticket #3893).

class `sage.rings.contfrac.`**`ContinuedFractionField`**

Bases: `sage.structure.unique_representation.UniqueRepresentation`, `sage.rings.ring.Field`

The field of rational implemented as continued fraction.

The code here is deprecated since in all situations it is better to use `QQ`.

See also:

`continued_fraction()`

EXAMPLES:

```
sage: CFF
QQ as continued fractions
sage: CFF([0,1,3,2])
[0; 1, 3, 2]
sage: CFF(133/25)
```

```
[5; 3, 8]

sage: CFF.category()
Category of fields
```

The continued fraction field inherits from the base class `sage.rings.ring.Field`. However it was initialised as such only since trac ticket trac ticket #11900:

```
sage: CFF.category()
Category of fields
```

class Element (*x1*, *x2=None*)

Bases: `sage.rings.continued_fraction.ContinuedFraction_periodic`, `sage.structure.element.FieldElement`

A continued fraction of a rational number.

EXAMPLES:

```
sage: CFF(1/3)
[0; 3]
sage: CFF([1,2,3])
[1; 2, 3]
```

`ContinuedFractionField.`**an_element**()

Returns a continued fraction.

EXAMPLES:

```
sage: CFF.an_element()
[-1; 2, 3]
```

`ContinuedFractionField.`**characteristic**()

Return 0, since the continued fraction field has characteristic 0.

EXAMPLES:

```
sage: c = CFF.characteristic(); c
0
sage: parent(c)
Integer Ring
```

`ContinuedFractionField.`**is_exact**()

Return True.

EXAMPLES:

```
sage: CFF.is_exact()
True
```

`ContinuedFractionField.`**is_field**(*proof=True*)

Return True.

EXAMPLES:

```
sage: CFF.is_field()
True
```

`ContinuedFractionField.`**is_finite**()

Return False, since the continued fraction field is not finite.

EXAMPLES:

```
sage: CFF.is_finite()
False
```

`ContinuedFractionField.`**`order`**`()`

EXAMPLES:

```
sage: CFF.order()
+Infinity
```

`ContinuedFractionField.`**`random_element`**`(*args, **kwds)`

Return a somewhat random continued fraction (the result is either finite or ultimately periodic).

INPUT:

- `args`, `kwds` - arguments passed to `QQ.random_element`

EXAMPLES:

```
sage: CFF.random_element() # random
[0; 4, 7]
```

`ContinuedFractionField.`**`some_elements`**`()`

Return some continued fractions.

EXAMPLES:

```
sage: CFF.some_elements()
([0], [1], [1], [-1; 2], [3; 1, 2, 3])
```

CHAPTER

EIGHTEEN

INTEGER FACTORIZATION FUNCTIONS

AUTHORS:

- Andre Apitzsch (2011-01-13): initial version

`sage.rings.factorint.aurifeuillian(n, m, F=None, check=True)`

Return the Aurifeuillian factors $F_n^\pm(m^2n)$.

This is based off Theorem 3 of [Brent93].

INPUT:

- `n` – integer
- `m` – integer
- `F` – integer (default: `None`)
- `check` – boolean (default: `True`)

OUTPUT:

A list of factors.

EXAMPLES:

```
sage: from sage.rings.factorint import aurifeuillian
sage: aurifeuillian(2,2)
[5, 13]
sage: aurifeuillian(2,2^5)
[1985, 2113]
sage: aurifeuillian(5,3)
[1471, 2851]
sage: aurifeuillian(15,1)
[19231, 142111]
sage: aurifeuillian(12,3)
Traceback (most recent call last):
...
ValueError: n has to be square-free
sage: aurifeuillian(1,2)
Traceback (most recent call last):
...
ValueError: n has to be greater than 1
sage: aurifeuillian(2,0)
Traceback (most recent call last):
...
ValueError: m has to be positive
```

Note: There is no need to set F. It's only for increasing speed of `factor_aurifeuillian()`.

REFERENCES:

`sage.rings.factorint.`**`factor_aurifeuillian`**`(`*n*, *check=True*`)`

Return Aurifeuillian factors of n if $n = x^{(2k-1)x} \pm 1$ (where the sign is '-' if x = 1 mod 4, and '+' otherwise) else n

INPUT:

- `n` – integer

OUTPUT:

List of factors of n found by Aurifeuillian factorization.

EXAMPLES:

```
sage: from sage.rings.factorint import factor_aurifeuillian as fa
sage: fa(2^6+1)
[5, 13]
sage: fa(2^58+1)
[536838145, 536903681]
sage: fa(3^3+1)
[4, 1, 7]
sage: fa(5^5-1)
[4, 11, 71]
sage: prod(_) == 5^5-1
True
sage: fa(2^4+1)
[17]
sage: fa((6^2*3)^3+1)
[109, 91, 127]
```

TESTS:

```
sage: for n in [2,3,5,6,30,31,33]:
....:     for m in [8,96,109201283]:
....:         s = -1 if n % 4 == 1 else 1
....:         y = (m^2*n)^n + s
....:         F = fa(y)
....:         assert(len(F) > 0 and prod(F) == y)
```

REFERENCES:

- http://mathworld.wolfram.com/AurifeuilleanFactorization.html
- [Brent93] Theorem 3

`sage.rings.factorint.`**`factor_cunningham`**`(`*m*, *proof=None*`)`

Return factorization of self obtained using trial division for all primes in the so called Cunningham table. This is efficient if self has some factors of type $b^n + 1$ or $b^n - 1$, with b in $\{2, 3, 5, 6, 7, 10, 11, 12\}$.

You need to install an optional package to use this method, this can be done with the following command line: `sage -i cunningham_tables`.

INPUT:

- `proof` – bool (default: `None`); whether or not to prove primality of each factor, this is only for factors not in the Cunningham table

EXAMPLES:

```
sage: from sage.rings.factorint import factor_cunningham
sage: factor_cunningham(2^257-1) # optional - cunningham
535006138814359 * 1155685395246619182673033 * 374550598501810936581776630096313181393
sage: factor_cunningham((3^101+1)*(2^60).next_prime(),proof=False) # optional - cunningham
2^2 * 379963 * 1152921504606847009 * 1017291527198723292208309354658785077827527
```

sage.rings.factorint.**factor_trial_division**(*m*, *limit='LONG_MAX'*)

Return partial factorization of self obtained using trial division for all primes up to limit, where limit must fit in a C signed long.

INPUT:

•`limit` – integer (default: `LONG_MAX`) that fits in a C signed long

EXAMPLES:

```
sage: from sage.rings.factorint import factor_trial_division
sage: n = 920384092842390423848290348203948092384082349082
sage: factor_trial_division(n, 1000)
2 * 11 * 41835640583745019265831379463815822381094652231
sage: factor_trial_division(n, 2000)
2 * 11 * 1531 * 27325696005058797691594630609938486205809701
```

TESTS:

Test that trac ticket #13692 is solved:

```
sage: from sage.rings.factorint import factor_trial_division
sage: list(factor_trial_division(8))
[(2, 3)]
```

sage.rings.factorint.**factor_using_pari**(*n*, *int_=False*, *debug_level=0*, *proof=None*)

Factors this (positive) integer using PARI.

This method returns a list of pairs, not a `Factorization` object. The first element of each pair is the factor, of type `Integer` if `int_` is `False` or `int` otherwise, the second element is the positive exponent, of type `int`.

INPUT:

•`int_` – (default: `False`), whether the factors are of type `int` instead of `Integer`

•`debug_level` – (default: 0), debug level of the call to PARI

•`proof` – (default: `None`), whether the factors are required to be proven prime; if `None`, the global default is used

OUTPUT:

A list of pairs.

EXAMPLES:

```
sage: factor(-2**72 + 3, algorithm='pari')   # indirect doctest
-1 * 83 * 131 * 294971519 * 1472414939
```

CHAPTER

NINETEEN

BASIC ARITHMETIC WITH C-INTEGERS.

class `sage.rings.fast_arith.`**`arith_int`**

Bases: `object`

`gcd_int`(*a*, *b*)

`inverse_mod_int`(*a*, *m*)

`rational_recon_int`(*a*, *m*)

Rational reconstruction of a modulo m.

`xgcd_int`(*a*, *b*)

class `sage.rings.fast_arith.`**`arith_llong`**

Bases: `object`

`gcd_longlong`(*a*, *b*)

`inverse_mod_longlong`(*a*, *m*)

`rational_recon_longlong`(*a*, *m*)

Rational reconstruction of a modulo m.

`sage.rings.fast_arith.`**`prime_range`**(*start*, *stop=None*, *algorithm='pari_primes'*, *py_ints=False*)

List of all primes between start and stop-1, inclusive. If the second argument is omitted, returns the primes up to the first argument.

This function is closely related to (and can use) the primes iterator. Use algorithm "pari_primes" when both start and stop are not too large, since in all cases this function makes a table of primes up to stop. If both are large, use algorithm "pari_isprime" instead.

Algorithm "pari_primes" is faster for most input, but crashes for larger input. Algorithm "pari_isprime" is slower but will work for much larger input.

INPUT:

- `start` – lower bound
- `stop` – upper bound
- `algorithm` – string, one of:
  - **"pari_primes": Uses PARI's primes function. Generates all primes up to stop.** Depends on PARI's primepi function.
  - **"pari_isprime": Uses a mod 2 wheel and PARI's isprime function by calling** the primes iterator.
- `py_ints` – boolean (default False), return Python ints rather than Sage Integers (faster)

EXAMPLES:

```
sage: prime_range(10)
[2, 3, 5, 7]
sage: prime_range(7)
[2, 3, 5]
sage: prime_range(2000,2020)
[2003, 2011, 2017]
sage: prime_range(2,2)
[]
sage: prime_range(2,3)
[2]
sage: prime_range(5,10)
[5, 7]
sage: prime_range(-100,10,"pari_isprime")
[2, 3, 5, 7]
sage: prime_range(2,2,algorithm="pari_isprime")
[]
sage: prime_range(10**16,10**16+100,"pari_isprime")
[10000000000000061, 10000000000000069, 10000000000000079, 10000000000000099]
sage: prime_range(10**30,10**30+100,"pari_isprime")
[1000000000000000000000000000057, 1000000000000000000000000000099]
sage: type(prime_range(8)[0])
<type 'sage.rings.integer.Integer'>
sage: type(prime_range(8,algorithm="pari_isprime")[0])
<type 'sage.rings.integer.Integer'>
```

TESTS:

```
sage: prime_range(-1)
[]
sage: L = prime_range(25000,2500000)
sage: len(L)
180310
sage: L[-10:]
[2499923, 2499941, 2499943, 2499947, 2499949, 2499953, 2499967, 2499983, 2499989, 2499997]
```

A non-trivial range without primes:

```
sage: prime_range(4652360, 4652400)
[]
```

AUTHORS:

- William Stein (original version)
- Craig Citro (rewrote for massive speedup)
- Kevin Stueve (added primes iterator option) 2010-10-16
- Robert Bradshaw (speedup using Pari prime table, py_ints option)

CHAPTER

TWENTY

MISCELLANEOUS UTILITIES

sage.rings.misc.**composite_field**(K, L)

Return a canonical field that contains both K and L, if possible. Otherwise, raise a ValueError.

INPUT: K – field L – field

OUTPUT: field

EXAMPLES: sage: composite_field(QQ,QQbar) Algebraic Field sage: composite_field(QQ,QQ[sqrt(2)]) Number Field in sqrt2 with defining polynomial x^2 - 2 sage: composite_field(QQ,QQ) Rational Field sage: composite_field(QQ,GF(7)) Traceback (most recent call last): ... ValueError: unable to find a common field

CHAPTER

TWENTYONE

MONOMIALS

sage.rings.monomials.**monomials**(v, n)

Given two lists v and n, of exactly the same length, return all monomials in the elements of v, where variable i (i.e., v[i]) in the monomial appears to degree strictly less than n[i].

INPUT:

- v – list of ring elements
- n – list of integers

EXAMPLES:

```
sage: monomials([x], [3])
[1, x, x^2]
sage: R.<x,y,z> = QQ[]
sage: monomials([x,y], [5,5])
[1, y, y^2, y^3, y^4, x, x*y, x*y^2, x*y^3, x*y^4, x^2, x^2*y, x^2*y^2, x^2*y^3, x^2*y^4, x^3, x
sage: monomials([x,y,z], [2,3,2])
[1, z, y, y*z, y^2, y^2*z, x, x*z, x*y, x*y*z, x*y^2, x*y^2*z]
```

CHAPTER

TWENTYTWO

ABSTRACT BASE CLASS FOR COMMUTATIVE ALGEBRAS

sage.rings.commutative_algebra.**is_CommutativeAlgebra**(*x*)

Check to see if x is a CommutativeAlgebra.

EXAMPLES:

```
sage: sage.rings.commutative_algebra.is_CommutativeAlgebra(sage.rings.ring.CommutativeAlgebra(ZZ
True
```

CHAPTER

TWENTYTHREE

BASE CLASS FOR COMMUTATIVE ALGEBRA ELEMENTS

CHAPTER

TWENTYFOUR

ABSTRACT BASE CLASS FOR COMMUTATIVE RINGS

`sage.rings.commutative_ring.is_CommutativeRing(R)`

Check to see if `R` is a `CommutativeRing`.

EXAMPLES:

```
sage: sage.rings.commutative_ring.is_CommutativeRing(ZZ)
True
```

CHAPTER

TWENTYFIVE

BASE CLASS FOR COMMUTATIVE RING ELEMENTS

sage.rings.commutative_ring_element.**is_CommutativeRingElement**(*x*)

Check to see if x is a CommutativeRingElement.

EXAMPLES:

```
sage: sage.rings.commutative_ring_element.is_CommutativeRingElement(ZZ(2))
True
```

CHAPTER

TWENTYSIX

BASE CLASS FOR DEDEKIND DOMAINS

sage.rings.dedekind_domain.**is_DedekindDomain**(*R*)

Check to see if R is a DedekindDomain.

EXAMPLES:

```
sage: sage.rings.dedekind_domain.is_DedekindDomain(DedekindDomain(QQ))
True
```

CHAPTER TWENTYSEVEN

BASE CLASS FOR DEDEKIND DOMAIN ELEMENTS

sage.rings.dedekind_domain_element.**is_DedekindDomainElement**(*x*)

Check to see if x is a DedekindDomainElement.

EXAMPLES:

```
sage: sage.rings.dedekind_domain_element.is_DedekindDomainElement(DedekindDomainElement(QQ))
True
```

CHAPTER

TWENTYEIGHT

ABSTRACT BASE CLASS FOR EUCLIDEAN DOMAINS

sage.rings.euclidean_domain.**is_EuclideanDomain**(*R*)

Check to see if R is a EuclideanDomain.

EXAMPLES:

```
sage: sage.rings.euclidean_domain.is_EuclideanDomain(EuclideanDomain(ZZ))
True
```

CHAPTER TWENTYNINE

BASE CLASS FOR EUCLIDEAN DOMAIN ELEMENTS

`sage.rings.euclidean_domain_element.`**`is_EuclideanDomainElement`**`(x)`

Check to see if `x` is a `EuclideanDomainElement`.

EXAMPLES:

```
sage: sage.rings.euclidean_domain_element.is_EuclideanDomainElement(EuclideanDomainElement(ZZ))
True
```

CHAPTER

THIRTY

ABSTRACT BASE CLASS FOR FIELDS

`sage.rings.field.is_PrimeField(R)`

Determine if `R` is a field that is equal to its own prime subfield.

INPUT:

- `R` – a ring or field

OUTPUT:

- `True` if R is $\mathbf{Q}$ or a finite field $\mathbf{F}_p$ for p prime, `False` otherwise.

EXAMPLES:

```
sage: import sage.rings.field
doctest:...: DeprecationWarning: the module sage.rings.field is deprecated and will be removed
See http://trac.sagemath.org/18108 for details.
sage: sage.rings.field.is_PrimeField(QQ)
True
sage: sage.rings.field.is_PrimeField(GF(7))
True
sage: sage.rings.field.is_PrimeField(GF(7^2,'t'))
False
```

CHAPTER

THIRTYONE

BASE CLASS FOR FIELD ELEMENTS

`sage.rings.field_element.is_FieldElement(x)`

Check to see if `R` is a `FieldElement`.

EXAMPLES:

```
sage: sage.rings.field_element.is_FieldElement(QQ(2))
True
```

CHAPTER

THIRTYTWO

ABSTRACT BASE CLASS FOR INTEGRAL DOMAINS

sage.rings.integral_domain.**is_IntegralDomain**(*R*)

Check if R is an instance of IntegralDomain.

EXAMPLES:

```
sage: sage.rings.integral_domain.is_IntegralDomain(QQ)
True
sage: sage.rings.integral_domain.is_IntegralDomain(ZZ)
True
```

CHAPTER

THIRTYTHREE

BASE CLASS FOR INTEGRAL DOMAIN ELEMENTS

sage.rings.integral_domain_element.**is_IntegralDomainElement**(*x*)

Check if x is an element of IntegralDomainElement.

EXAMPLES:

```
sage: sage.rings.integral_domain_element.is_IntegralDomainElement(ZZ(2))
True
```

CHAPTER

THIRTYFOUR

ABSTRACT BASE CLASS FOR PRINCIPAL IDEAL DOMAINS

sage.rings.principal_ideal_domain.**is_PrincipalIdealDomain**(*R*)

Check if R is a PrincipalIdealDomain.

EXAMPLES:

```
sage: sage.rings.principal_ideal_domain.is_PrincipalIdealDomain(ZZ)
True
sage: R.<x,y> = QQ[]
sage: sage.rings.principal_ideal_domain.is_PrincipalIdealDomain(R)
False
```

CHAPTER

THIRTYFIVE

BASE CLASS FOR PRINCIPAL IDEAL DOMAIN ELEMENTS

sage.rings.principal_ideal_domain_element.**is_PrincipalIdealDomainElement**(*x*)

Check to see if x is a PrincipalIdealDomainElement.

EXAMPLES:

```
sage: sage.rings.principal_ideal_domain_element.is_PrincipalIdealDomainElement(ZZ(2))
True
```

CHAPTER

THIRTYSIX

BASE CLASS FOR RING ELEMENTS

CHAPTER THIRTYSEVEN

INDICES AND TABLES

[AtiMac] Atiyah and Macdonald, "Introduction to commutative algebra", Addison-Wesley, 1969.

[Ho72] E. Horowitz, "Algorithms for Rational Function Arithmetic Operations", Annual ACM Symposium on Theory of Computing, Proceedings of the Fourth Annual ACM Symposium on Theory of Computing, pp. 108–118, 1972

[WpInvariantTheory] http://en.wikipedia.org/wiki/Glossary_of_invariant_theory

[WpBinaryForm] http://en.wikipedia.org/wiki/Invariant_of_a_binary_form

[WpTernaryCubic] http://en.wikipedia.org/wiki/Ternary_cubic

[Salmon] G. Salmon: "A Treatise on the Analytic Geometry of Three Dimensions", section on "Invariants and Covariants of Systems of Quadrics".

[Salmon2] G. Salmon: A Treatise on Conic Sections, Section on "Invariants and Covariants of Systems of Conics", Art. 388 (a).

[Brent93] Richard P. Brent. *On computing factors of cyclotomic polynomials*. Mathematics of Computation. **61** (1993). No. 203. pp 131-149. Arxiv 1004.5466v1. http://www.jstor.org/stable/2152941

r

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

X

Z